



# Desarrollo de Entornos Web

---

<b>Curso</b>	Desarrollo de Entornos Web (2351)
<b>Formato</b>	Manual de curso
<b>Autor Institucional</b>	Cibertec
<b>Páginas</b>	312 p.
<b>Elaborador</b>	López Aragón, Dámaso
<b>Revisor de Contenidos</b>	Morales Flores, Gustavo

# Índice

Presentación	5
Red de contenidos	6

## UNIDAD DE APRENDIZAJE 1: PROGRAMACIÓN BÁSICA EN LENGUAJE JAVASCRIPT

<b>1.1 Tema 1 : Fundamentos de JavaScript</b>	<b>8</b>
1.1.1 : Introducción al JavaScript	8
1.1.2 : Elementos de un programa en JavaScript	8
1.1.2.1 : Entradas y salidas: alert, prompt y confirm	8
1.1.2.2 : Escritura de datos: document.write	11
1.1.2.3 : Ámbito de las variables	11
1.1.2.4 : Estructuras de control: condicionales, selectivas y repetitivas, sintaxis	12

## UNIDAD DE APRENDIZAJE 2: EL MODELO DE OBJETO DE DOCUMENTO (DOM)

<b>2.1 Tema 2 : Modelo DOM</b>	<b>41</b>
2.1.1 : Definición	41
2.1.2 : Árbol de nodos	41
2.1.3 : Tipos de nodos	44
2.1.4 : Acceso directo a los nodos: getelementbyid, getelementsbytagname, getelementsbyclassname, queryselector	45
2.1.5 : Manejo de estilos de los elementos	47
2.1.6 : Acceso directo a los atributos: getattribute, setattribute	53

## UNIDAD DE APRENDIZAJE 3: FUNCIONES Y EVENTOS

<b>3.1 Tema 3 : Manejo de funciones</b>	<b>76</b>
3.1.1 : Introducción	76
3.1.2 : Sintaxis general	76
3.1.3 : Funciones y argumentos	77
3.1.4 : Funciones con return	79
3.1.5 : Manejo del setTimeout() y setInterval()	83
<b>3.2 Tema 4 : Eventos</b>	<b>104</b>
3.2.1 : Introducción	104
3.2.2 : Tipos de eventos	105
3.2.3 : Manejadores de eventos	107
3.2.4 : Manejo de listener	110

## UNIDAD DE APRENDIZAJE 4: MANEJO DE ARREGLOS

<b>4.1 Tema 5 : Arreglos</b>	<b>140</b>
------------------------------	------------

4..1.1 : Introducción	140
4.1.2 : Definición de arreglos, creación	140
4.1.3 : La clase array	141
4.1.4 Trabajando con arreglo de elementos	145
<b>4.2 Tema 6 : ChildNodes</b>	<b>158</b>
4.2.1 : Trabajando con childNodes	158
4.2.2 : Métodos para acceder a nodos hijos	159
4.2.3 : Métodos para agregar o quitar nodos	159

**UNIDAD DE APRENDIZAJE 5: FORMULARIOS**

<b>5.1 Tema 7 : Formularios</b>	<b>179</b>
5.1.1 : El objeto form	179
5.1.2 : Objetos de un formulario	180
5.1.3 : Propiedades y eventos de un formulario	181
5.1.4 : Acceso a los elementos de un formulario	182
<b>5.2 Tema 8 : Expresiones regulares</b>	<b>204</b>
5.2.1 : Introducción	204
5.2.2 : Definición de expresiones regulares	204
5.2.2.1 : Creación de expresiones	204
5.2.2.2 : Manejo de caracteres especiales	207

**UNIDAD DE APRENDIZAJE 6: REACT JS**

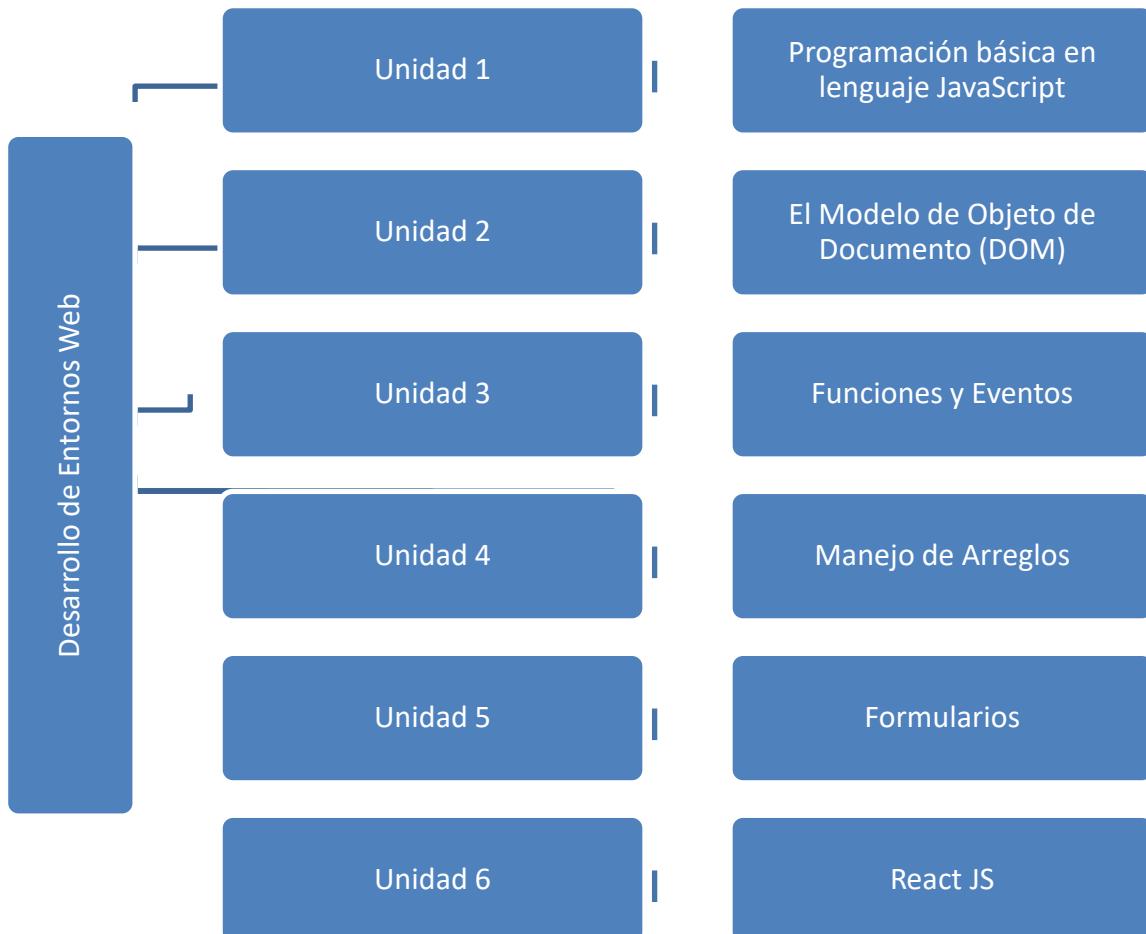
<b>6.1 Tema 9 : Introducción</b>	<b>232</b>
6.1.1 : Definición	232
6.1.1.1 : Componentes	232
6.1.1.2 : Virtual DOM	233
6.1.2 : Configurando e instalando el entorno react	234
6.1.3 : Nuestra primera aplicación con react desde un entorno vacío	235
<b>6.2 Tema 10 : Trabajando con react</b>	<b>254</b>
6.2.1 : Componentes react con createclass	256
6.2.2 : Componentes react mediante clases ES6	257
6.2.3 : Propiedades y estados en componentes react	259
<b>6.3 Tema 11 : Ciclo de vida y Eventos en React</b>	<b>279</b>
6.3.1 : Ciclo de vida de los componentes	279
6.3.2 : Eventos en React	281
6.3.3 : Condicionales en templates JSX de react	285
6.3.4 : Creación de repeticiones en templates JSX con react	286
6.3.5 : Extendiendo componentes con mixins	287
<b>Bibliografía</b>	<b>312</b>

# Presentación

Desarrollo de entornos web es un curso que pertenece a la línea técnica y se dicta en las carreras Computación e Informática, Administración y Sistemas, Redes y Electrónica. Brinda a los alumnos un conjunto de aplicativos, como editores de textos para HTML5, CSS, JavaScript, así como el framework React JS, para el diseño y desarrollo de sitios web con aplicaciones multimedia y validación de formularios.

El curso es eminentemente práctico y consiste en el diseño de páginas web y programación avanzada con JavaScript para realizar operaciones, cálculos y validaciones en las mismas. En primer lugar, se inicia con el lenguaje JavaScript básico y avanzado con aplicaciones en el desarrollo de entornos web. Luego, estudiaremos React JS que es un producto que sirve como base para la programación avanzada de aplicaciones de tipo SPA.

# Red de contenidos





# PROGRAMACIÓN BÁSICA EN JAVASCRIPT

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno, con el lenguaje JavaScript, diseña programas, incorporados en una página del Sitio Web y para validar formularios.

## TEMARIO

- 1.1 Tema 1 : Fundamentos de JavaScript**
  - 1.1.1 : Introducción al JavaScript
  - 1.1.2 : Elementos de un programa en JavaScript
  - 1.1.2.1 : Entradas y salidas: alert, prompt y confirm
  - 1.1.2.2 : Escritura de datos: document.write
  - 1.1.2.3 : Ámbito de las variables
  - 1.1.2.4 : Estructuras de control: condicionales, selectivas y repetitivas, sintaxis

## ACTIVIDADES PROPUESTAS

- Los alumnos aprender a manejar las entradas y salidas.
- Los alumnos desarrollan actividades con las estructuras de control.

## 1.1. FUNDAMENTOS DE JAVASCRIPT

### 1.1.1 Introducción al JavaScript

JavaScript es un lenguaje interpretado orientado a las páginas web, con una sintaxis semejante a la del lenguaje Java.

El lenguaje fue inventado por Brendan Eich en la empresa Netscape Communications, que es la que fabricó los primeros navegadores de Internet comerciales.

Apareció por primera vez en el producto de Netscape llamado Netscape Navigator 2.0.

Se utiliza en páginas web HTML, para realizar tareas y operaciones en el marco de la aplicación cliente.

Los autores inicialmente lo llamaron Mocha y más tarde LiveScript, pero fue rebautizado como JavaScript en un anuncio conjunto entre Sun Microsystems y Netscape, el 4 de diciembre de 1995.

### 1.1.2 Elementos de un programa en JavaScript

El lenguaje JavaScript utiliza una sintaxis parecida a la de Java y admite:

- Estructuras de selección y de repetición, como **if...else**, **for** y **do...while**.
- Se utilizan llaves (**{}**) para delimitar los bloques de instrucciones.
- El lenguaje admite varios tipos de datos, como String, Number, Boolean, Object y Array.
- Admite las características de fecha mejoradas, las funciones trigonométricas y las expresiones regulares.
- JavaScript usa prototipos en vez de clases. Puede definir un objeto creando una función constructora.
- JavaScript no declara explícitamente los tipos de datos de las variables. En muchos casos, JavaScript realiza las conversiones automáticamente cuando es necesario.

#### 1.1.2.1 . Entradas y salidas: alert, prompt, confirm

##### La instrucción alert(cadena)

Se utiliza para mostrar mensajes o datos en cuadros de diálogos predefinidos. El siguiente ejemplo muestra el valor del atributo href de la etiqueta <a> llamada enlace.

##### Código JavaScript:

```
var enlace = document.getElementById("enlace");
alert(enlace.href); // muestra http://www...com
```

##### Código HTML:

```
<a id="enlace" href="http://www...com">Enlace</a>
```

El siguiente ejemplo obtiene el valor de la propiedad margin de la imagen:

**Código JavaScript:**

```
var imagen = document.getElementById("imagen");
alert(imagen.style.margin);
```

**Código HTML:**

```

```

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

**Código JavaScript:**

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.style.fontWeight); // muestra "bold"
```

**Código HTML:**

```
<p id="parrafo" style="font-weight: bold;">...</p>
```

**La instrucción prompt(cadena)**

Se utiliza para pedir al usuario que introduzca un texto en una ventana modal. Veamos un ejemplo:

**Código JavaScript:**

```
<script>
// Pedimos al usuario que introduzca su nombre
var nombre = prompt("Introduzca su nombre");
// Mostramos texto concatenado con el nombre ingresado
alert("Hola "+nombre);
</script>
```

**La instrucción confirm**

Se utiliza para visualizar por pantalla un mensaje acompañado de los botones ‘Aceptar’ y ‘Cancelar’. Este tipo de mensajes se muestran cuando se requiere, por parte del usuario, la aceptación o cancelación de una acción.

Los valores devueltos serán ‘true’ en caso de aceptar y ‘false’ en caso de cancelar. Veamos un ejemplo:

El siguiente código muestra un mensaje con dos botones Aceptar y Cancelar:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
<script>
    var a=1,b=5;
```

```

var opcion=confirm(a+" es mayor que "+b+"?");
if(opcion)alert("hizo clic en Aceptar");
else alert("hizo clic en Cancelar");
</script>
</head>
<body>
</body>
</html>

```

Cada navegador muestra el cuadro de alerta de diferentes formas. Por ejemplo, del código anterior:

Internet Explorer muestra así el alert



Figura 1: Alert  
Fuente.- Elaboración Propia

Chrome muestra de la siguiente manera

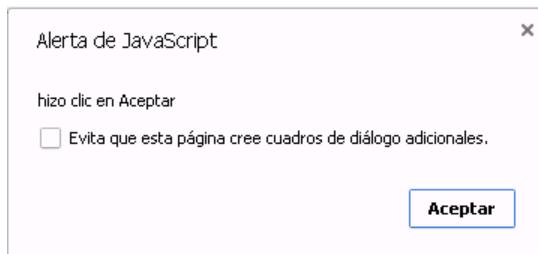


Figura 2: Alert  
Fuente.- Elaboración Propia

Firefox muestra así el alert:



Figura 3: Alert  
Fuente.- Elaboración Propia

### 1.1.2.2 .Escritura de datos: document.write

Se utiliza para escribir un texto en el documento html actual. El primer ejemplo escribe en la página el texto 'Hola Mundo', el segundo el resultado de la variable screen.width que devuelve el ancho de la pantalla en pixeles y el tercero agrupa texto y la variable.

```
<script>
document.write('Hola Mundo<br>')
document.write(+screen.width+'<br>')
document.write('Ancho de la pantalla: '+screen.width+' pixeles')
</script>
```

Se mostrará en la página lo siguiente:

Hola Mundo

1024

Ancho de la pantalla: 1024 pixeles

Cuando se ejecuta `document.write()`, después de cargar la página, se borrará todo el contenido de la página web y solo mostrará el texto dentro de ella.

En la actualidad, el `document.write()` es cada vez menos utilizado y en su reemplazo utilizamos `innerHTML` y `appendChild` por agregar elementos en forma dinámica.

#### 1.1.2.3. Ámbito de las variables

El ámbito de una variable (llamado "scope" en inglés) es la área de programación en la que se define la variable. JavaScript define dos ámbitos para las variables: global y local.

Una variable local solamente está definida dentro de la función. Cualquier instrucción que se encuentre dentro de la función puede hacer uso de la variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable mensaje.

##### Ejemplo de variable local.

```
function creaMensaje() {
  var mensaje = "Mensaje de prueba";
  alert(mensaje);
}
```

Una variable global, se encuentra definida en cualquier punto del programa, la cual puede ser utilizada por cualquier función definada en el área de programación.

##### Ejemplo de variable global

```
var mensaje = "Mensaje de prueba";
function muestraMensaje() {
  alert(mensaje);
}
```

##### Ámbito de una variable: var.

En el enfoque tradicional de JavaScript, es decir, cuando se utiliza la palabra clave **var** es para declarar variables. Existen dos ámbitos principales: ámbito global y ámbito a nivel de función.

En el ejemplo siguiente, declare la variable **x** con un valor inicial 3, mientras dentro de una función, la variable **x** inicializada en ella estarán en el ámbito de la propia función.

```

var x=3;
function Mostrar(){
    var x=5;
    alert("Valor de x en la funcion:"+x);
}

Mostrar();
alert("Valor de x fuera de la funcion"+x);

```

#### Ámbito de una variable: let.

En las versiones modernas de JavaScript (ES6 o ECMAScript 2015) o posteriores, se introduce la palabra clave **let** en sustitución de var. Con ella, en lugar de utilizar los ámbitos globales y a nivel de función (var), utilizamos los ámbitos clásicos de programación: ámbito global y ámbito local.

#### 1.1.2.4. Estructuras de control: condicionales, selectivas, repetitivas, sintaxis

##### Estructura condicional

La estructura más utilizada en JavaScript y en la mayoría de los lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```

if(condicion) {
    ...
}

```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de las instrucciones del script.

Ejemplo:

```

var mensaje = true;
if(mensaje) {
    alert("Hola Mundo");
}

```

En el ejemplo anterior, el mensaje sí que se muestra al usuario ya que la variable mensaje tiene un valor de true y, por tanto, el programa entra dentro del bloque de instrucciones del if.

El ejemplo se podría reescribir también como:

```

var mensaje = true;
if(mensaje == true) {
    alert("Hola Mundo");
}

```

En este caso, la condición es una comparación entre el valor de la variable mensaje y el valor true. Como los dos valores coinciden, la igualdad se cumple y por tanto la condición es cierta, su valor es true y se ejecutan las instrucciones contenidas en ese bloque del if.

La comparación del ejemplo anterior suele ser el origen de muchos errores de programación, al confundir los operadores == y =. Las comparaciones siempre se realizan con el operador ==, ya que el operador = solamente asigna valores:

```
var mensaje = true;
// Se comparan los dos valores
if(mensaje == false) {
    ...
}
// Error - Se asigna el valor "false" a la variable
if(mensaje = false) {
    ...
}
```

La condición que controla el if() puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;
if(!mostrado) {
    alert("Es la primera vez que se muestra el mensaje");
}
```

Los operadores AND (&&) y OR (||) permiten encadenar varias condiciones simples para construir condiciones complejas:

```
var mostrado = false;
var mensaje = true;

if(!mostrado &&mensaje) {
    alert("Es la primera vez que se muestra el mensaje");
}
```

La condición anterior está formada por una operación AND sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación AND. De esta forma, como el valor de mostrado es false, ¡el valor !mostrado sería true. Como la variable mensaje vale true, ¡el resultado de !mostrado &&mensaje sería igual a true && true, por lo que el resultado final de la condición del if() sería true y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del if().

### Estructura if else

En ocasiones, las decisiones que se deben realizar no son del tipo "*si se cumple la condición, hazlo; si no se cumple, no hagas nada*". Normalmente las condiciones suelen ser del tipo "*si se cumple esta condición, hazlo; si no se cumple, haz esto otro*".

Para este segundo tipo de decisiones, existe una variante de la estructura if llamada if...else. Su definición formal es la siguiente:

```
if(condicion) {
    ...
}
else {
```

```
...  
{
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }. Ejemplo:

```
var edad = 18;  
  
if(edad >= 18) {  
    alert("Eres mayor de edad");  
}  
else {  
    alert("Todavía eres menor de edad");  
}
```

Si el valor de la variable edad es mayor o igual que el valor numérico 18, la condición del if() se cumple y, por tanto, se ejecutan sus instrucciones y se muestra el mensaje "Eres mayor de edad". Sin embargo, cuando el valor de la variable edad no es igual o mayor que 18, la condición del if() no se cumple, por lo que automáticamente se ejecutan todas las instrucciones del bloque else { }. En este caso, se mostraría el mensaje "Todavía eres menor de edad".

El siguiente ejemplo compara variables de tipo cadena de texto:

```
var nombre = "";  
  
if(nombre == "") {  
    alert("No has ingresado tu nombre");  
}  
else {  
    alert("Nombre grabado correctamente");  
}
```

La condición del if() anterior se construye mediante el operador ==, que es el que se emplea para comparar dos valores (no confundir con el operador = que se utiliza para asignar valores). En el ejemplo anterior, si la cadena de texto almacenada en la variable nombre es vacía (es decir, es igual a "") se muestra el mensaje definido en el if(). En otro caso, se muestra el mensaje definido en el bloque else { }.

### Estructura if else if

La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

```
if(edad < 12) {  
    alert("Todavía eres muy pequeño");  
}  
else if(edad < 19) {  
    alert("Eres un adolescente");  
}  
else if(edad < 35) {  
    alert("Aun sigues siendo joven");  
}  
else {
```

```
    alert("Piensa en cuidarte un poco más");
}
```

No es obligatorio que la combinación de estructuras if...else acabe con la instrucción else, ya que puede terminar con una instrucción de tipo if().

### Estructura selectiva

A diferencia del if else, el switch case nos permite tener varias opciones o simplificar un poco el código. En el siguiente ejemplo dependiendo del día de la semana imprime un texto diferente.

```
<html>
<body>
<script type="text/JavaScript">
var a = new Date();
dia=a.getDay();
switch (dia)
{
case 5:
    document.write("<b>Viernes social</b>");
    break;
case 6:
    document.write("<b>Sábado deportivo</b>");
    break;
case 0:
    document.write("<b>Domingo familiar</b>");
    break;
default:
    document.write(i"<b>añoro que llegue el fin de semana!</b>");
}
</script>

<p>En este script el Domingo=0, Lunes=1, Martes=2, etc.</p>

</body>
</html>
```

### Estructura Repetitiva

#### Estructura while

La estructura while permite crear bucles (instrucciones repetitivas) que se ejecutan una o más veces, dependiendo de la condición indicada. Su definición formal es:

```
while(condicion) {
    ...
}
```

El funcionamiento del bucle while se resume en: "mientras se cumpla la condición indicada, repite indefinidamente las instrucciones incluidas dentro del bucle".

Si la condición no se cumple ni siquiera la primera vez, el bucle no se ejecuta. Si la condición se cumple, se ejecutan las instrucciones una vez y se vuelve a comprobar la condición. Si se sigue

cumpliendo la condición, se vuelve a ejecutar el bucle y así se continúa hasta que la condición no se cumpla.

Evidentemente, las variables que controlan la condición deben modificarse dentro del propio bucle, ya que, de otra forma, la condición se cumpliría siempre y el bucle while se repetiría indefinidamente.

El siguiente ejemplo utiliza el bucle while para sumar todos los números menores o iguales que otro número:

```
var resultado = 0;
var numero = 100;
var i = 0;
while(i <= numero) {
    resultado += i;
    i++;
}
alert(resultado);
```

El programa debe sumar todos los números menores o igual que otro dado. Por ejemplo, si el número es 5, se debe calcular:  $1 + 2 + 3 + 4 + 5 = 15$

Este tipo de condiciones "*suma números mientras sean menores o iguales que otro número dado*" se resuelven muy fácilmente con los bucles tipo while, aunque también se podían resolver con bucles de tipo for.

En el ejemplo anterior, mientras se cumpla la condición, es decir, mientras que la variable i sea menor o igual que la variable número, se ejecutan las instrucciones del bucle.

Dentro del bucle se suma el valor de la variable i al resultado total (variable resultado) y se actualiza el valor de la variable i, que es la que controla la condición del bucle. Si no se actualiza el valor de la variable i, la ejecución del bucle continua infinitamente o hasta que el navegador permita al usuario detener el script.

### Estructura do while

El bucle de tipo do...while es muy similar al bucle while, salvo que en este caso **siempre** se ejecutan las instrucciones del bucle al menos la primera vez. Su definición formal es:

```
do {
    ...
} while(condicion);
```

De esta forma, como la condición se comprueba después de cada repetición, la primera vez siempre se ejecutan las instrucciones del bucle. Es importante no olvidar que después del while() se debe añadir el carácter ; (al contrario de lo que sucede con el bucle while simple).

Utilizando este bucle se puede calcular fácilmente la factorial de un número:

```

var resultado = 1;
var numero = 5;

do {
    resultado *= numero; // resultado = resultado * numero
    numero--;
} while(numero > 0);

alert(resultado);

```

En el código anterior, el resultado se multiplica en cada repetición por el valor de la variable número. Además, en cada repetición se decrementa el valor de esta variable numero. La condición del bucle do...while es que el valor de numero sea mayor que 0, ya que la factorial de un número multiplica todos los números menores o iguales que él mismo, pero hasta el número 1 (el factorial de 5 por ejemplo es  $5 \times 4 \times 3 \times 2 \times 1 = 120$ ).

Como en cada repetición se decrementa el valor de la variable número y la condición es que numero sea mayor que cero, en la repetición en la que numero valga 0, la condición ya no se cumple y el programa se sale del bucle do...while.

### Estructura for

La estructura for permite realizar bucles de manera sencilla. Su sintaxis es la siguiente:

```

for(inicializacion; condicion; actualizacion) {

    ...
}

```

La idea del funcionamiento de un bucle for es la siguiente: "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición".

- La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
- La "condición" es el único elemento que decide si continua o se detiene la repetición.
- La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

Ejemplo:

```

var mensaje = "Hola, estoy dentro de un bucle";

for(var i = 0; i < 5; i++) {
    document.write(mensaje);
}

```

La parte de la inicialización del bucle consiste en: var i = 0;

Por tanto, en primer lugar, se crea la variable i y se le asigna el valor de 0. Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización.

La zona de condición del bucle es: `i < 5`

Los bucles se siguen ejecutando mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple.

En este caso, mientras la variable `i` valga menos de 5 el bucle se ejecuta indefinidamente.

Como la variable `i` se ha inicializado a un valor de 0 y la condición para salir del bucle es que `i` sea menor que 5, si no se modifica el valor de `i` de alguna forma, el bucle se repetiría indefinidamente. Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle: `i++`.

En este caso, el valor de la variable `i` se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el `for`.

Así, durante la ejecución de la quinta repetición el valor de `i` será 4. Después de la quinta ejecución, se actualiza el valor de `i`, que ahora valdrá 5. Como la condición es que `i` sea menor que 5, la condición ya no se cumple y las instrucciones del `for` no se ejecutan una sexta vez.

Normalmente, la variable que controla los bucles `for` se llama `i`, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

Ejemplo:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];  
  
for(var i=0; i<7; i++) {  
    document.write(dias[i]);  
}
```

### Estructura `for in`

La estructura JavaScript `for in` nos permite recorrer una lista de elementos de una forma sencilla. JavaScript `for in` es una estructura en bucle que nos permite tratar los elementos indicados en la sentencia.

La estructura JavaScript `for in` tiene la siguiente sintaxis:

```
for (variable in objeto) {  
    // Acciones  
}
```

La estructura de control `for in` es muy sencilla de utilizar, pero tiene el inconveniente de que el número de repeticiones que se realizan sólo se pueden controlar mediante las variables definidas en la zona de actualización del bucle.

Las sentencias `break` y `continue`, permiten manipular el comportamiento normal de los bucles `for in` para detener el bucle o para saltarse algunas repeticiones. Concretamente, la sentencia `break` permite terminar de forma abrupta un bucle y la sentencia `continue` permite saltarse algunas repeticiones del bucle.

El siguiente ejemplo muestra el uso de la sentencia `break`:

```

var cadena = "Instituto Superior Tecnologico Privado Cibertec";
var letras = cadena.split("");
var resultado = "";

for(i in letras)
{
    if(letras[i] == 'a') {
        break;
    }
    else {
        resultado += letras[i];
    }
}
alert(resultado);
// muestra "InstitutoSuperior Tecnologico Priv"

```

Si el programa llega a una instrucción de tipo `break`, sale inmediatamente del bucle y continúa ejecutando el resto de las instrucciones que se encuentran fuera del bucle `for`. En el ejemplo anterior, se recorren todas las letras de una cadena de texto y cuando se encuentra con la primera letra "a", se detiene la ejecución del bucle `for`.

La utilidad de `break` es terminar la ejecución del bucle cuando una variable toma un determinado valor o cuando se cumple alguna condición.

En ocasiones, lo que se desea es saltarse alguna repetición del bucle cuando se dan algunas condiciones. Siguiendo con el ejemplo anterior, ahora se desea que el texto de salida elimine todas las letras "o" de la cadena de texto original:

```

var cadena = "Instituto Superior Tecnologico Privado Cibertec";
var letras = cadena.split("");
var resultado = "";

for(i in letras) {
    if(letras[i] == 'o') {
        continue;
    }
    else {
        resultado += letras[i];
    }
}
alert(resultado);
// muestra "Institut Superir Tecnlgic Privad Cibertec"

```

En este caso, cuando se encuentra una letra "a" no se termina el bucle, sino que no se ejecutan las instrucciones de esa repetición y se pasa directamente a la siguiente repetición del bucle `for`.

La utilidad de `continue` es que permite utilizar el bucle `for` para filtrar los resultados en función de algunas condiciones o cuando el valor de alguna variable coincide con un valor determinado.

## LABORATORIO 1

### Programación básica de JavaScript

Se pide diseñar una página web donde:

1. A través de una alerta visualice mensaje de bienvenida
2. Imprima el título: Visitando la gastronomía en el Perú
3. Imprima una imagen
4. Imprima el nombre del usuario ingresado por un prompt

#### 1. Creando el Sitio Web

Defina una carpeta llamado SitioWeb\_01, en ella crea las siguientes subcarpetas, tal como se muestra.

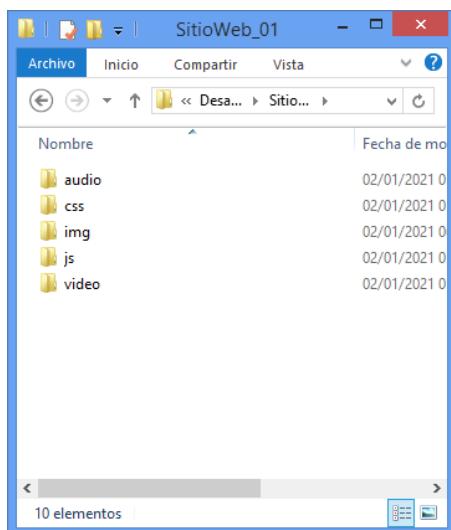


Figura 4 : Caso práctico  
Fuente.- Elaboración Propia

En la carpeta imágenes, agregar los archivos de extensión .jpg

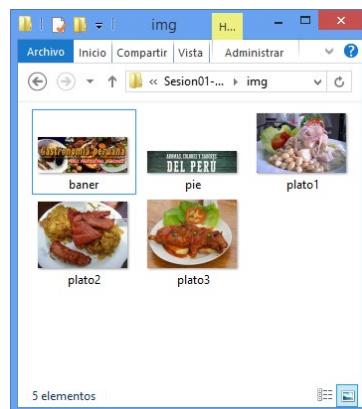


Figura 5: Caso práctico  
Fuente.- Elaboración Propia

## 2. Trabajando con Visual Studio Code

Para crear una página web, abrir el aplicativo Visual Studio Code.

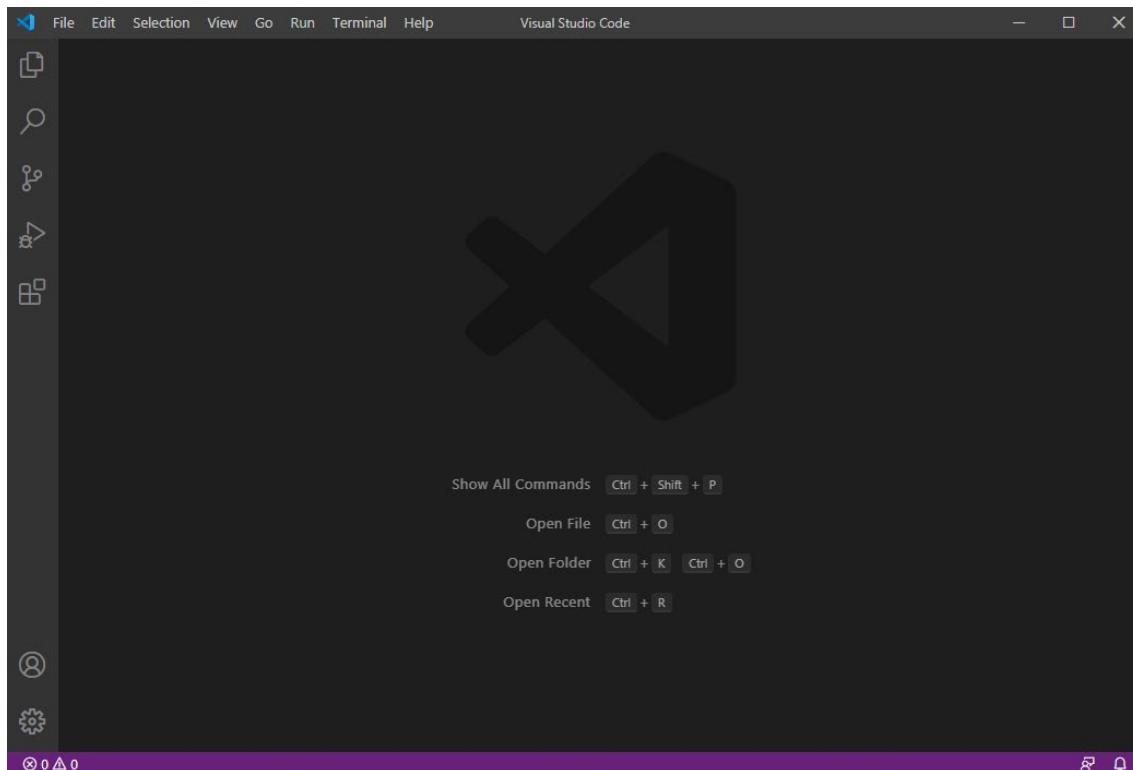


Figura 6 : Caso práctico  
Fuente.- Elaboración Propia

Para trabajar con la carpeta llamada Sesion01 en Visual Studio Code, arrastre la carpeta hacia la ventana del Visual, tal como se muestra; y luego soltarla.

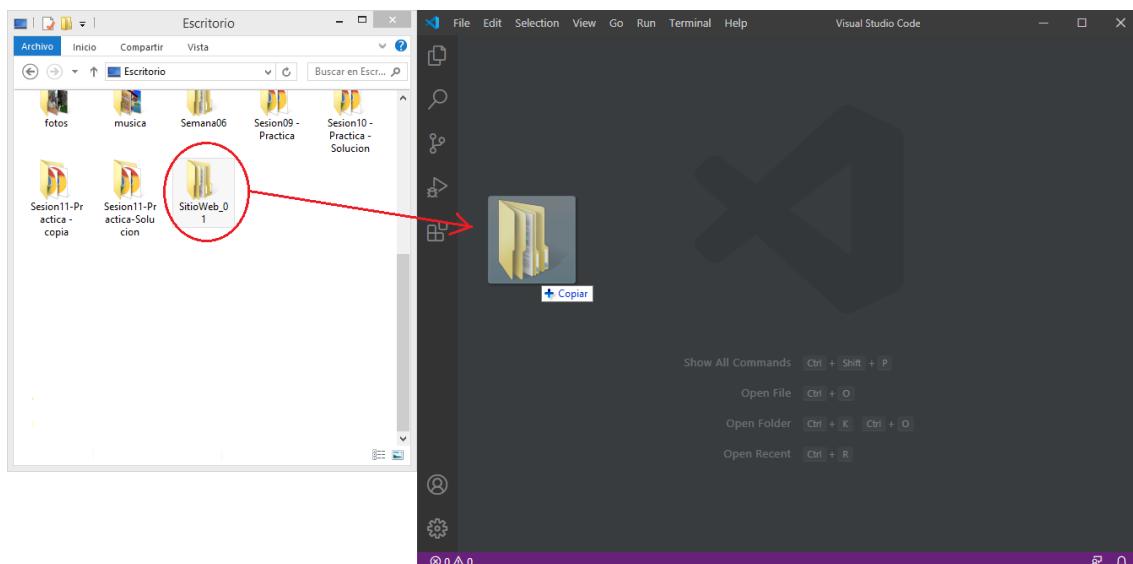


Figura 7: Caso práctico  
Fuente.- Elaboración Propia

Donde se visualiza la carpeta en la parte izquierda de la ventana del aplicativo

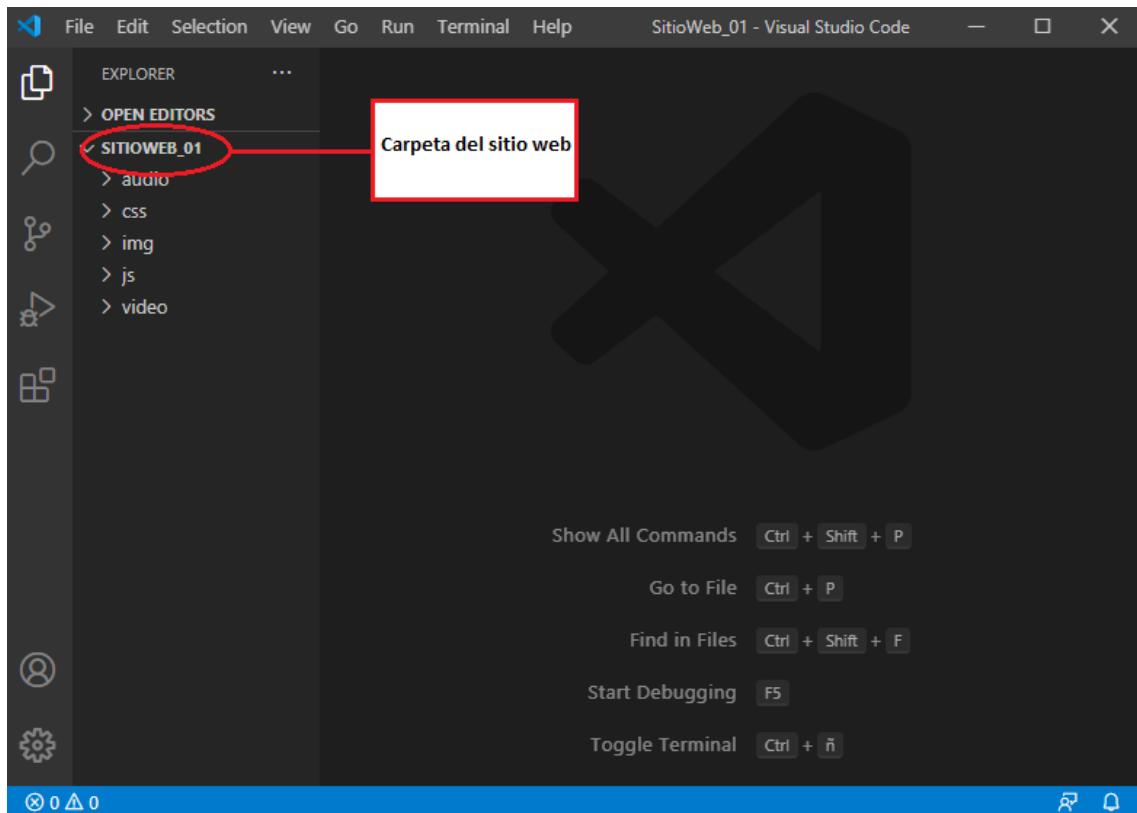


Figura 8: Caso práctico  
Fuente.- Elaboración Propia

### 3. Creando la página HTML

Para crear una página web, primero creamos un nuevo archivo: Desde la opción **File**, selecciona la opción **New File**, o simplemente utilizando la combinación de teclas Ctrl+N.

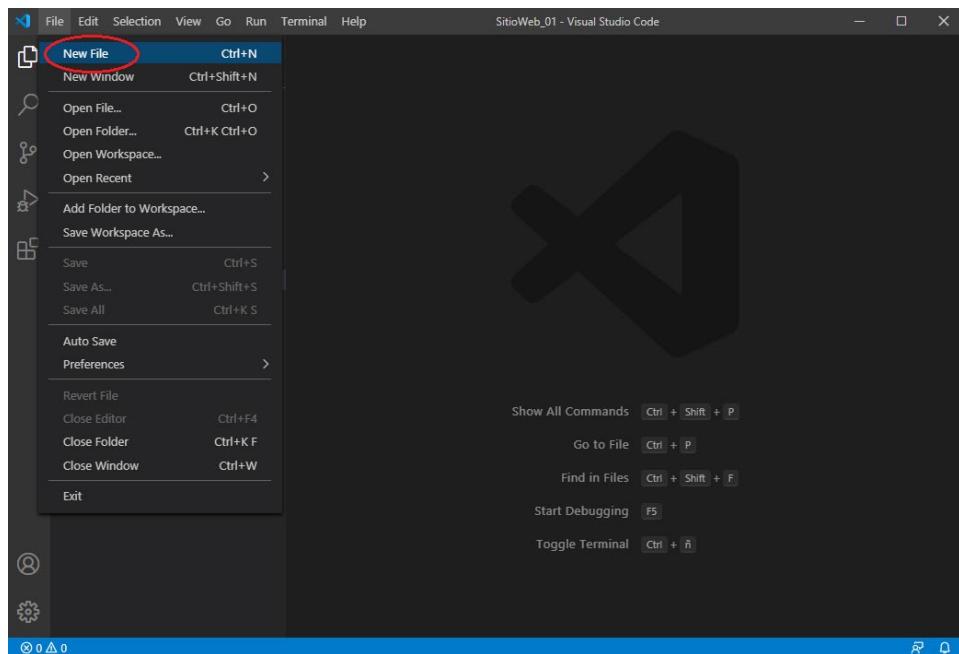


Figura 9: Caso práctico  
Fuente.- Elaboración Propia

Selecciona la opción, aparece un archivo con una etiqueta “Untitled-1”. A continuación, guardamos el archivo para que sea de tipo html.

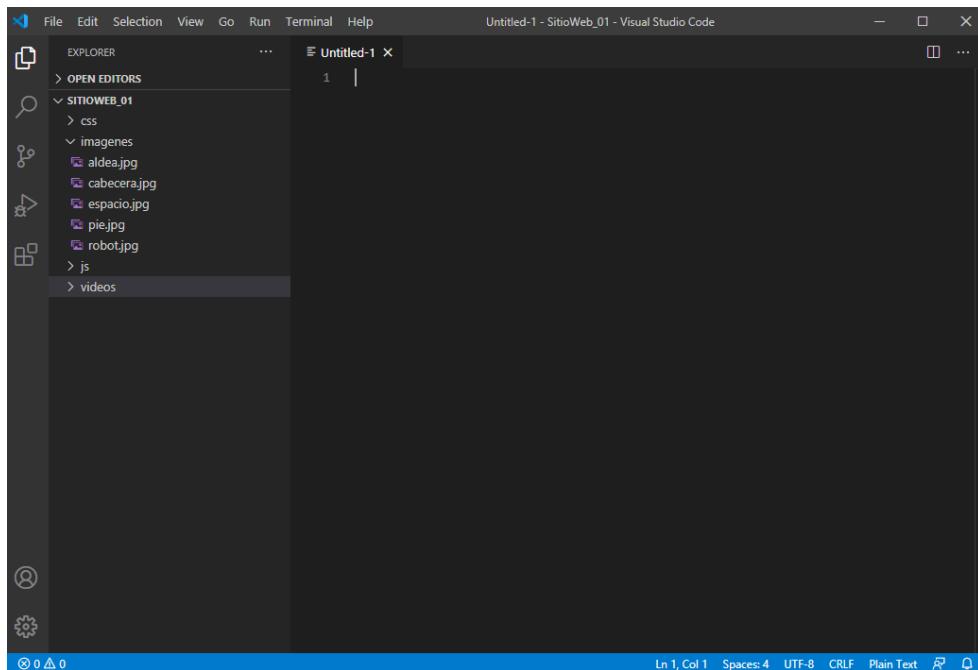


Figura 10: Caso práctico  
Fuente.- Elaboración Propia

Para guardar el archivo, desde la opción **File**, selecciona la opción **Save**, o también presionar la combinación de teclas Ctrl+S.

En la ventana Save As, asigne el nombre del archivo: pagina1; y selecciona el tipo de archivo el cual será HTML, tal como se muestra. Presionar el botón GUARDAR

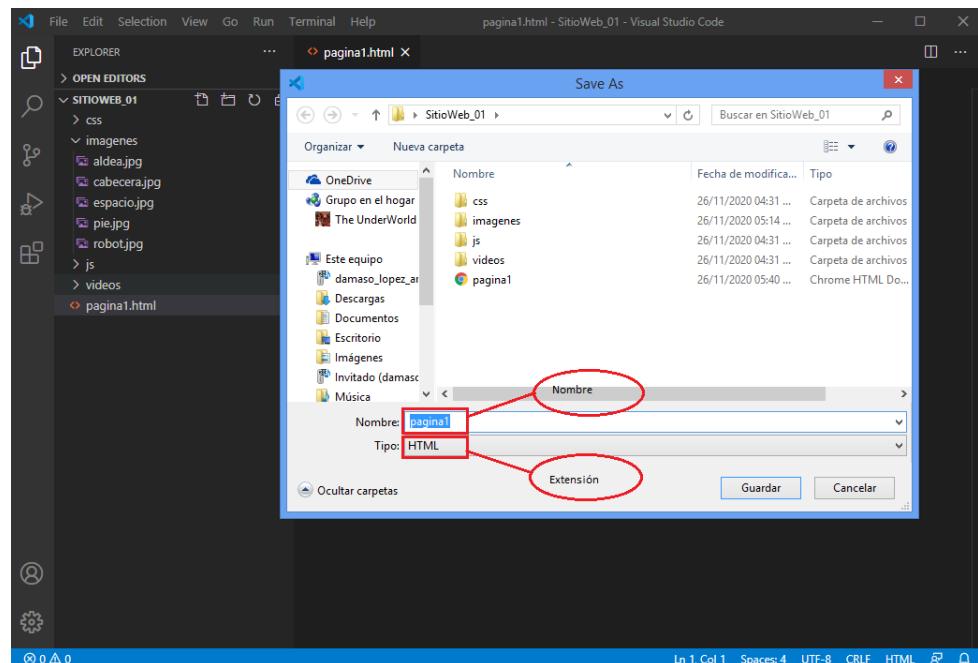


Figura 11: Caso práctico  
Fuente.- Elaboración Propia

En la página, ya creada, debemos agregar la estructura HTML. Para ello, escriba en la primera línea html, y selecciona de la lista html:5, y dar ENTER.

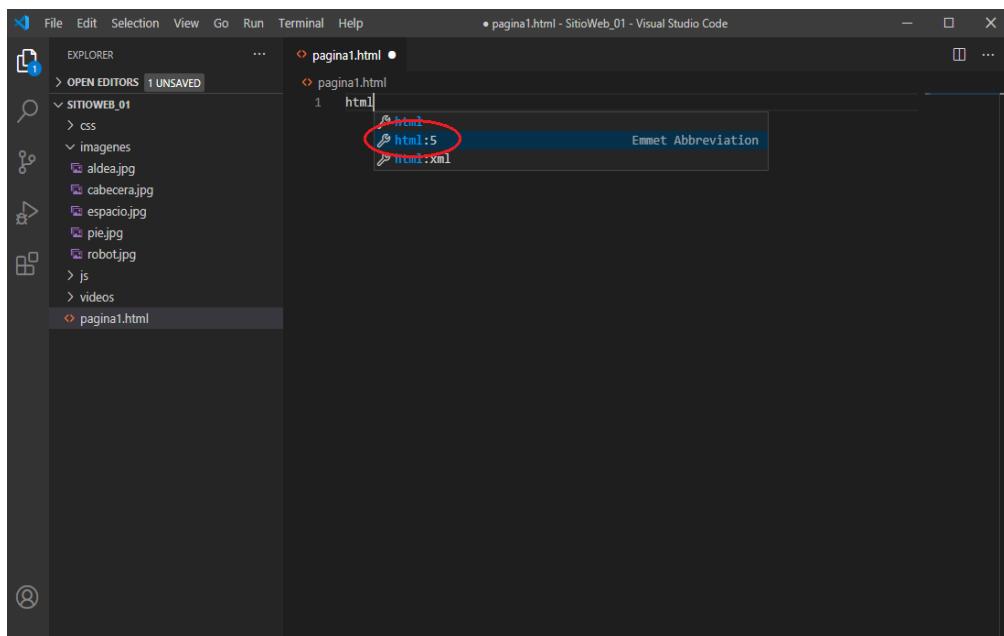


Figura 12: Caso práctico  
Fuente.- Elaboración Propia

Presionada la tecla ENTER, visualizamos la estructura del html, versión 5, tal como se muestra. A partir de aquí podemos diseñar la página web.

A screenshot of the Visual Studio Code interface, identical to Figure 12 but after pressing Enter. The code editor now displays the full HTML structure: "



", "", "", "", "", and "Document". The code editor has syntax highlighting and line numbers from 1 to 11.

Figura 13: Caso práctico  
Fuente.- Elaboración Propia

A continuación, agregamos un archivo CSS

1. Presione la combinación Ctrl + N (Nuevo archivo)
2. Para guardar presione Ctrl + S
3. Selecciona la carpeta CSS. asigne el nombre: estilos y el tipo de archivo CSS, tal como se muestra.
4. Presiona el botón ACEPTAR

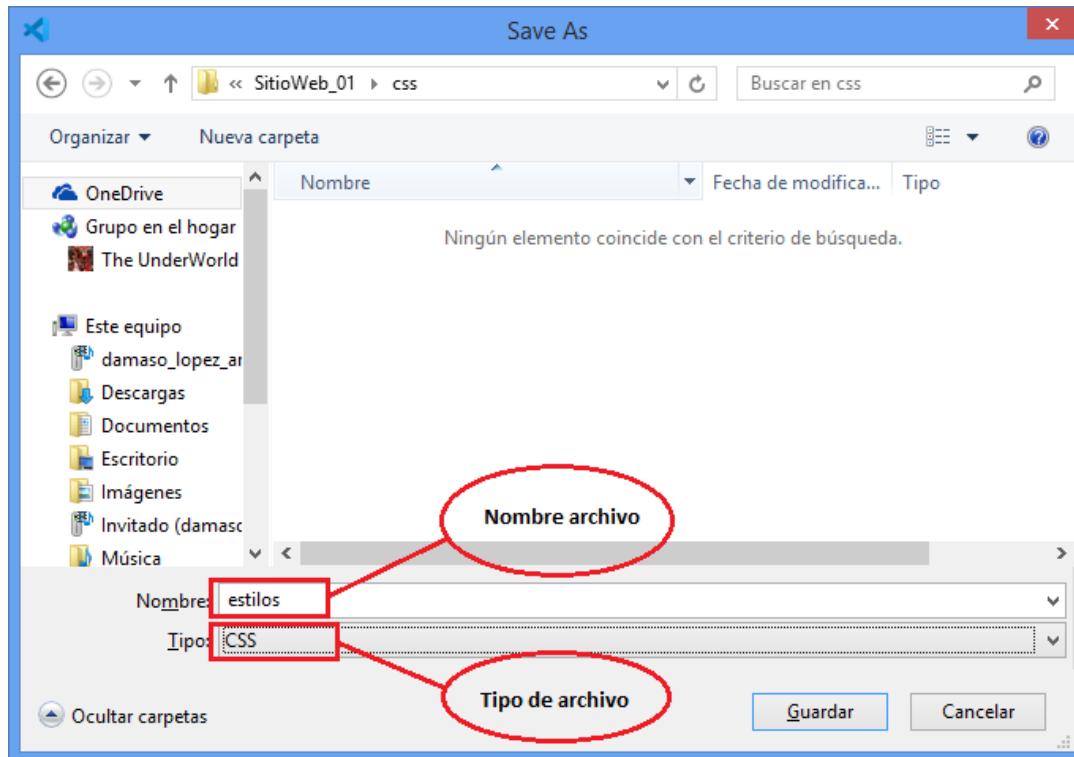


Figura 14: Caso práctico  
Fuente.- Elaboración Propia

Defina las reglas a los selectores de etiqueta que se agregarán en la página web, tal como se muestra. A continuación, guardar el contenido del archivo: Ctrl + S.

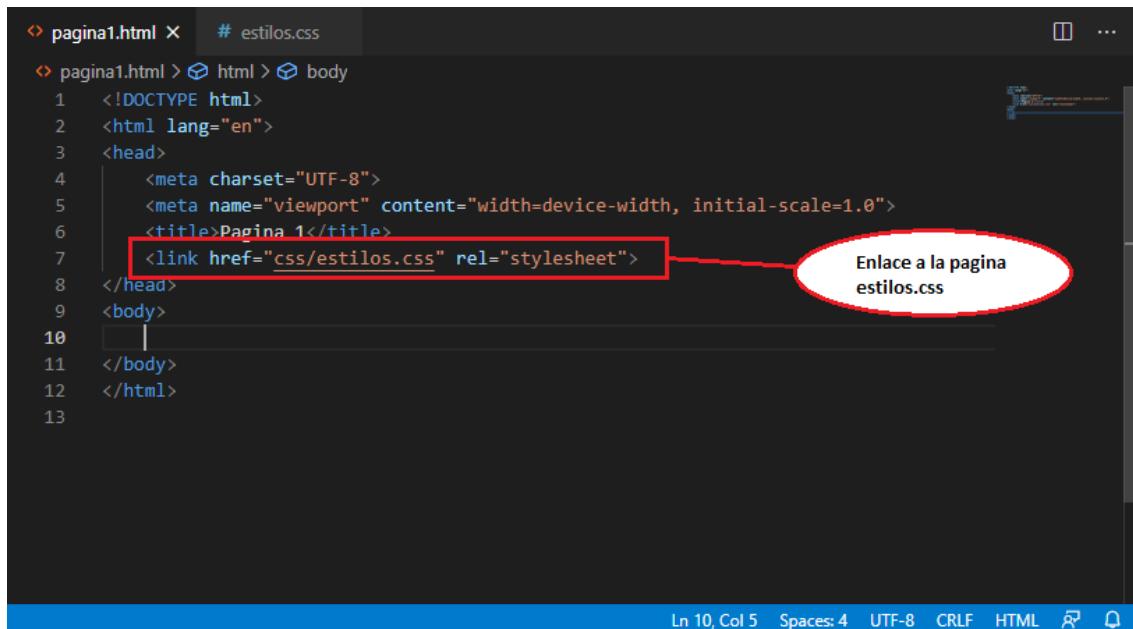
```
pagina1.html # estilos.css ...
css > # estilos.css > ...
1 h1 {
2     text-align: center;
3 }
4
5 img {
6     width: 100%;
7     height: 250px;
8 }
9
10 p {
11     width: 100%;
12     height: 20px;
13     float: left;
14 }
```

Ln 17, Col 1 Spaces: 4 UTF-8 CRLF CSS ⌂ ⌂

Figura 15: Caso práctico  
Fuente.- Elaboración Propia

#### 4. Diseño de la página HTML

Para iniciar el proceso, primero nos enlazamos al archivo estilos.css utilizando la etiqueta <link>, tal como se muestra.



```

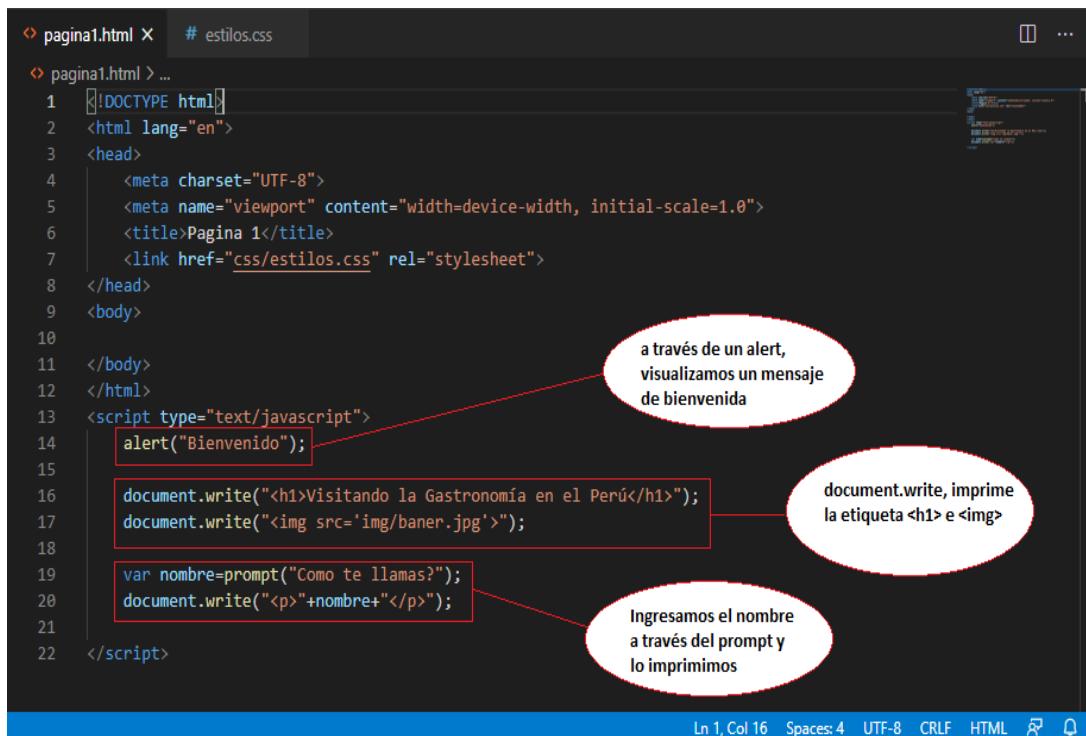
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pagina 1</title>
    <link href="css/estilos.css" rel="stylesheet">
  </head>
  <body>
  </body>
</html>

```

Figura 16 : Caso práctico

Fuente.- Elaboración Propia

Al final de la etiqueta <html>, agrega la etiqueta <script>, donde definimos las siguientes instrucciones la cuales se ejecutarán cuando se cargue la página a través del navegador.



```

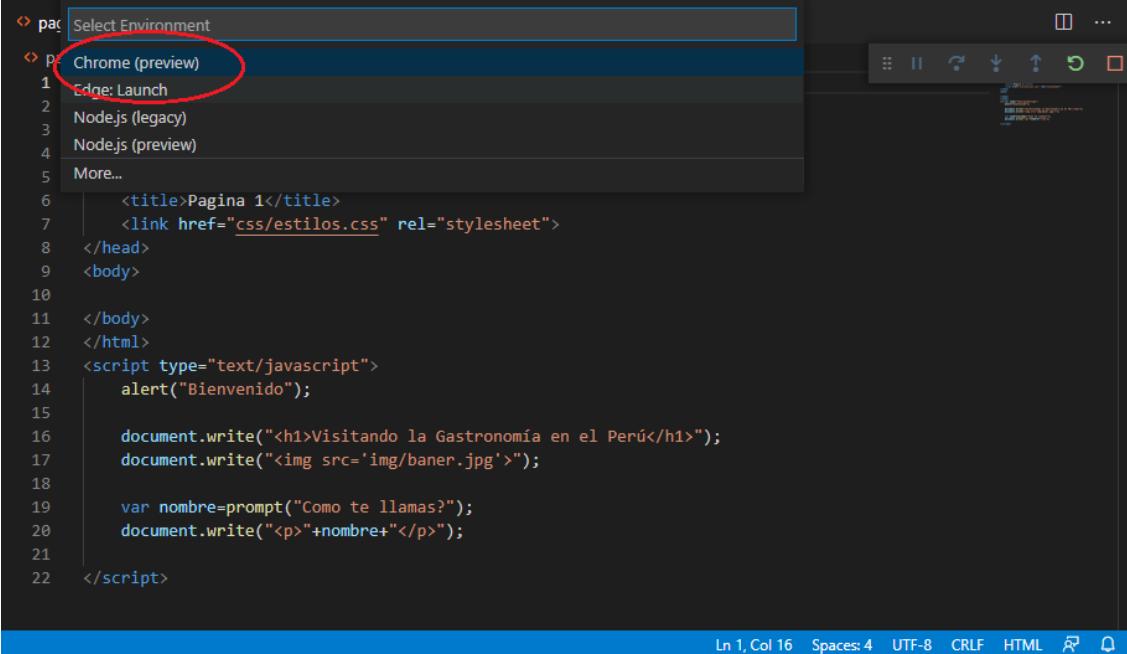
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pagina 1</title>
    <link href="css/estilos.css" rel="stylesheet">
  </head>
  <body>
  </body>
</html>
<script type="text/javascript">
  alert("Bienvenido");
  document.write("<h1>Visitando la Gastronomía en el Perú</h1>");
  document.write("<img src='img/baner.jpg'>");
  var nombre=prompt("Como te llamas?");
  document.write("<p>" + nombre + "</p>");
</script>

```

Figura 17: Caso práctico

Fuente.- Elaboración Propia

A continuación, presiona la tecla F5 y selecciona Chrome (preview), tal como se muestra.



```

    <!-- pag 1 -->
    Select Environment
    <--> P Chrome (preview)
    1 Edge: Launch
    2 Node.js (legacy)
    3 Node.js (preview)
    4 More...
    5
    6     <title>Pagina 1</title>
    7     <link href="css/estilos.css" rel="stylesheet">
    8 </head>
    9 <body>
    10 </body>
    11 </html>
    12 <script type="text/javascript">
    13     alert("Bienvenido");
    14
    15     document.write("<h1>Visitando la Gastronomía en el Perú</h1>");
    16     document.write("<img src='img/baner.jpg'>");
    17
    18     var nombre=prompt("Como te llamas?");
    19     document.write("<p>" +nombre+"</p>");
    20
    21 </script>
    22
  
```

Ln 1, Col 16 Spaces: 4 UTF-8 CRLF HTML ⚙️ 🗑️

Figura 18: Caso práctico  
Fuente.- Elaboración Propia

Al ejecutar la página, primero visualizamos el mensaje de bienvenida; después de ACEPTAR, visualizamos una ventana donde ingresamos un nombre, al presionar el botón ACEPTAR, visualizamos los elementos impresos por document.write()



Figura 19: Caso práctico  
Fuente.- Elaboración Propia

## LABORATORIO 2

### Programación básica de JavaScript

Se pide diseñar una página web donde , en un <table> visualice:

1. Imprima la fecha (el mes estará expresado en letras)
2. Imprima la hora

### Trabajando con el sitio web

Para este ejercicio, estamos trabajando con el sitio web del laboratorio 1.

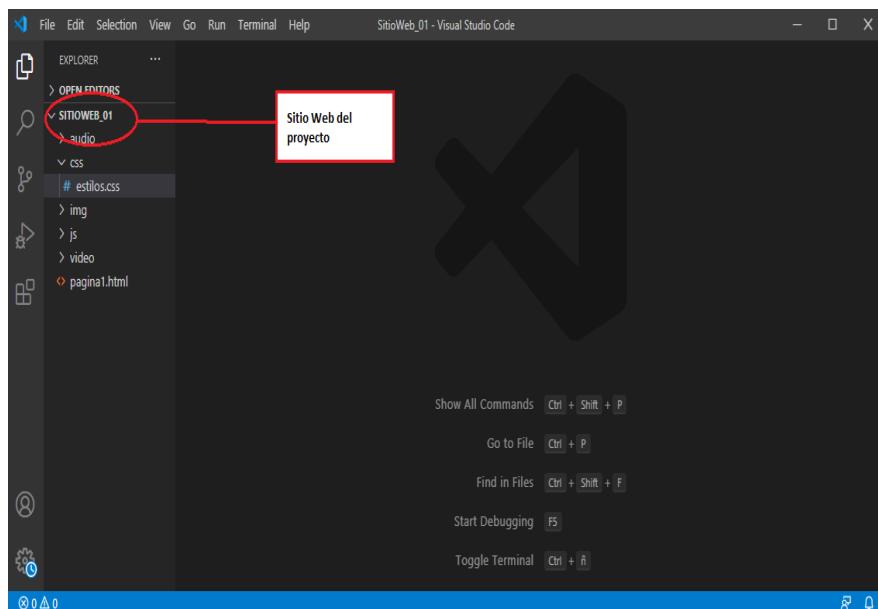


Figura 20: Caso práctico  
Fuente.- Elaboración Propia

A continuación, agregamos una página HTML (Ctrl + N) y al guardar (Ctrl + S), se llamará pagina2.HTML, ubicada en el carpeta raíz del sitio web, tal como se muestra.

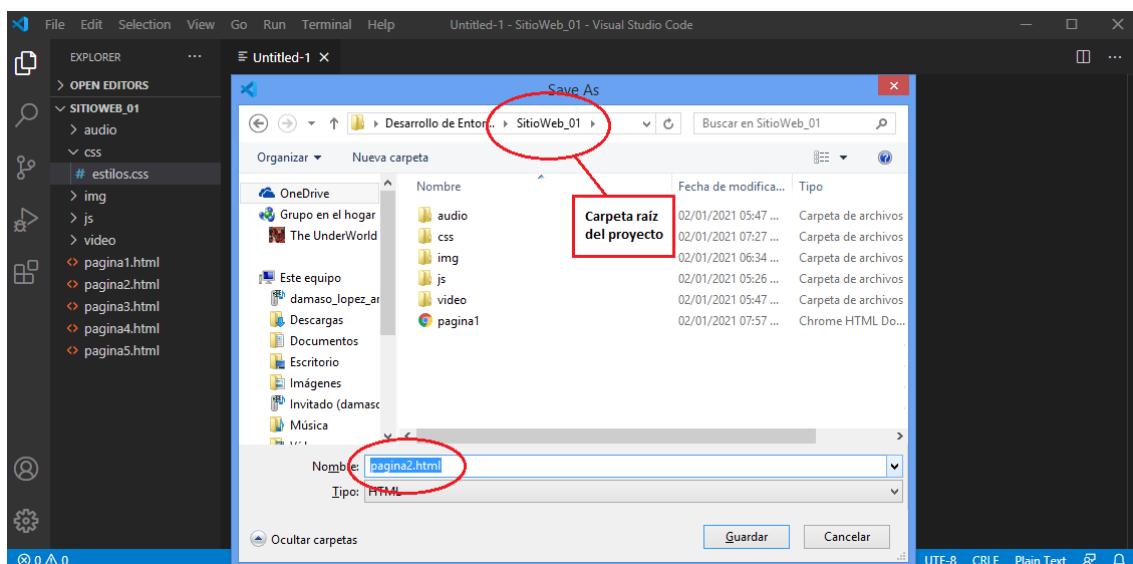


Figura 21: Caso práctico  
Fuente.- Elaboración Propia

En la página, ya creada, debemos agregar la estructura HTML. Para ello, escriba en la primera línea html, y selecciona de la lista html:5, y dar ENTER.

The screenshot shows a code editor interface with a dark theme. In the top left, there are two folder icons labeled 'pagina2.html'. Below them is a text input field containing '1 html'. A red circle highlights the 'html:5' option in a dropdown menu that appears when the input is focused. A red arrow points from a callout box containing the text 'Utilizando la abreviación Emmet, definimos las etiquetas de la página' to the 'html:5' option. To the right of the input field is a floating Emmet Abbreviation panel displaying the expanded HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
</body>
</html>
```

At the bottom of the editor window, status bars show 'Ln 1, Col 5', 'Tab Size: 4', 'UTF-8', 'CRLF', 'HTML', and icons for file operations.

Figura 22: Caso práctico  
Fuente.- Elaboración Propia

Visualizando la estructura de la página HTML, tal como se muestra en la figura.

The screenshot shows a code editor interface with a dark theme. In the top left, there are two folder icons labeled 'pagina2.html'. Below them is a text input field containing the expanded HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Document</title>
7
8 </head>
9 <body>
10
11 </body>
12 </html>
```

At the bottom of the editor window, status bars show 'Ln 7, Col 5', 'Tab Size: 4', 'UTF-8', 'CRLF', 'HTML', and icons for file operations.

Figura 23: Caso práctico  
Fuente.- Elaboración Propia

A continuación, creamos un nuevo archivo (Ctrl + N) y lo guardamos (Ctrl + S) en la carpeta JS, con el nombre de archivo1 y tipo JavaScript, tal como se muestra.

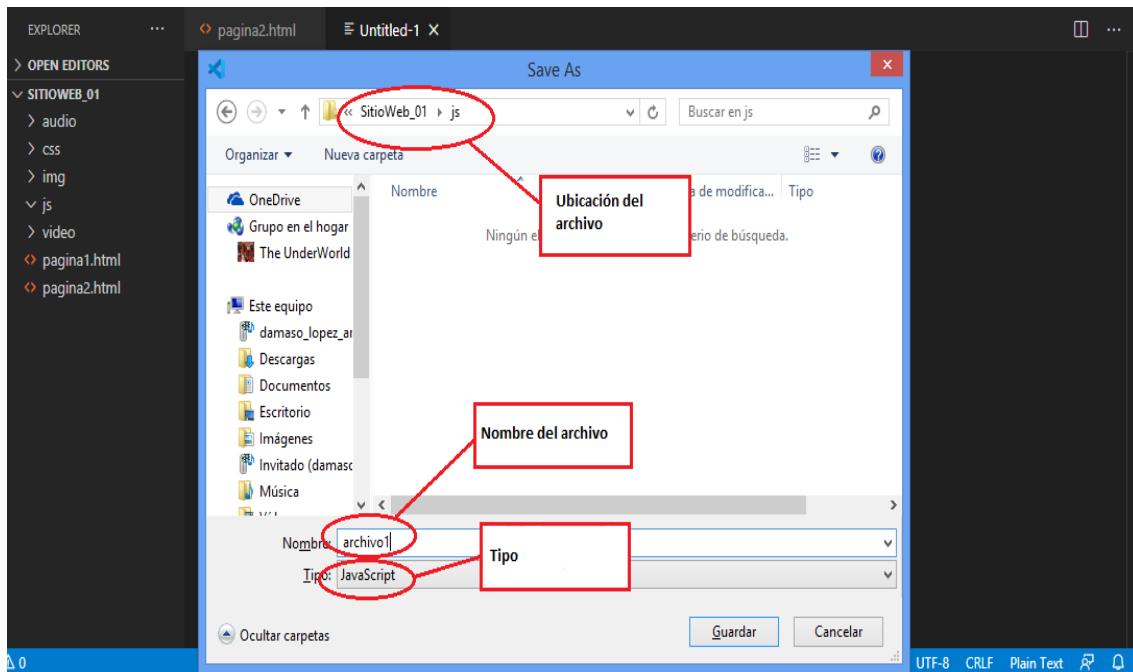


Figura 24: Caso práctico  
Fuente.- Elaboración Propia

### Archivo1.js

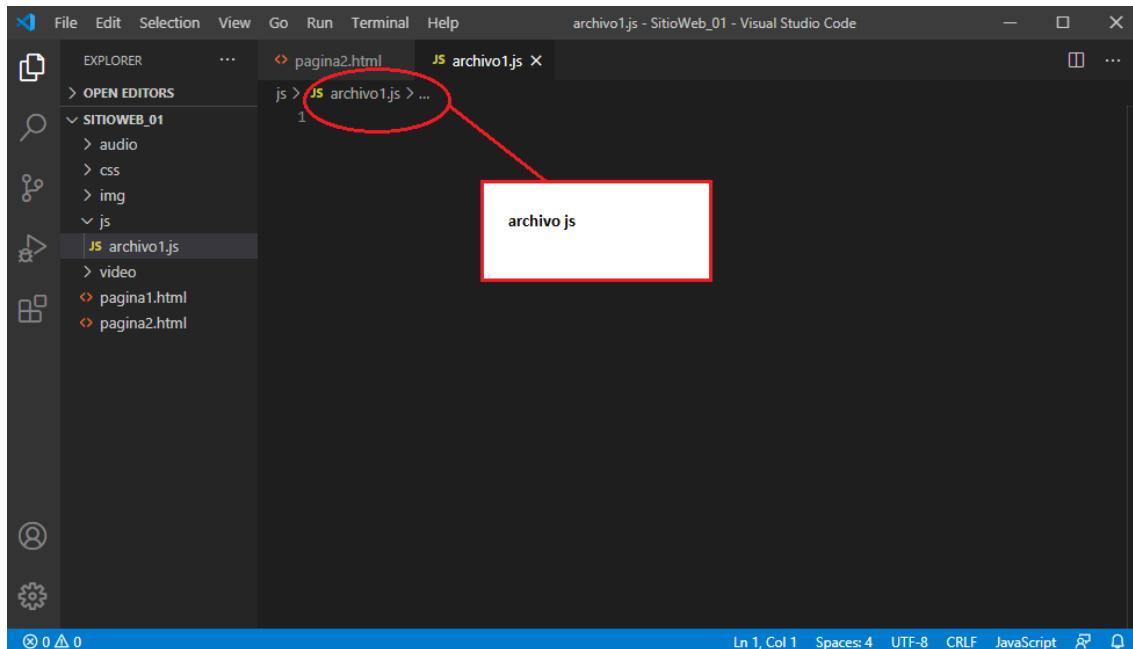
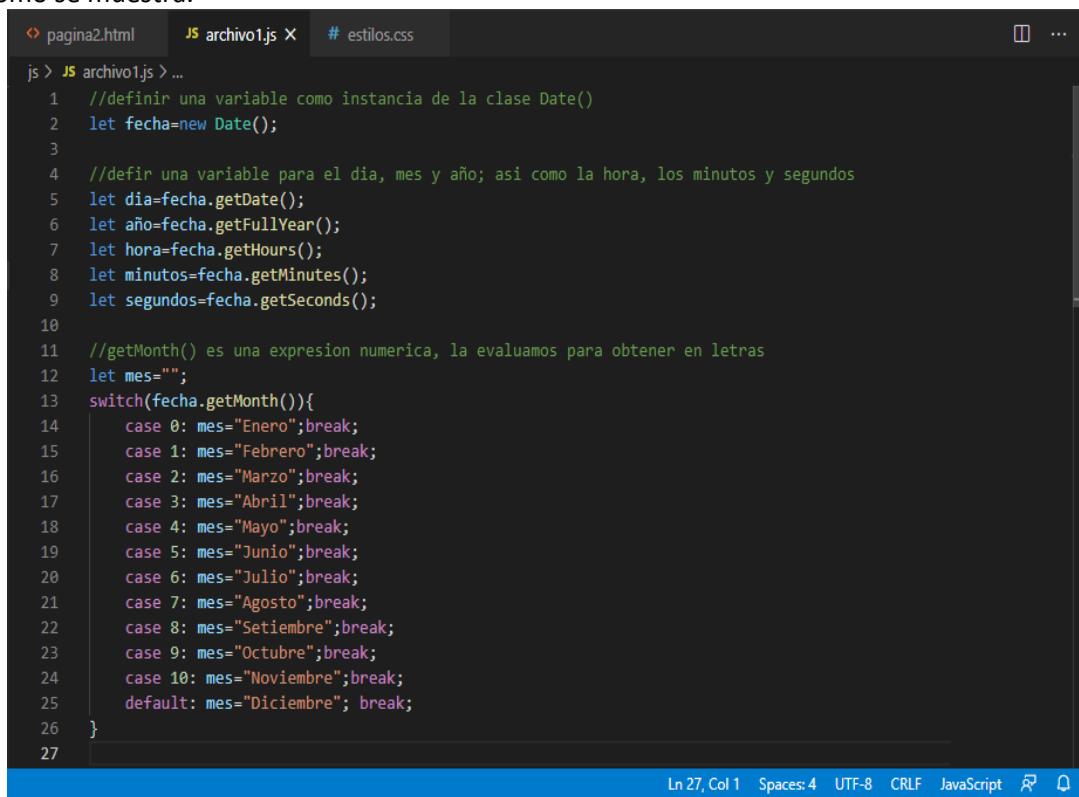


Figura 25: Caso práctico  
Fuente.- Elaboración Propia

En el archivo de JavaScript, defina las variables para almacenar los datos de la fecha y hora, tal como se muestra.



```

pagina2.html    JS archivo1.js # estilos.css
js > JS archivo1.js > ...
1 //definir una variable como instancia de la clase Date()
2 let fecha=new Date();
3
4 //definir una variable para el dia, mes y año; así como la hora, los minutos y segundos
5 let dia=fecha.getDate();
6 let año=fecha.getFullYear();
7 let hora=fecha.getHours();
8 let minutos=fecha.getMinutes();
9 let segundos=fecha.getSeconds();
10
11 //getMonth() es una expresión numérica, la evaluamos para obtener en letras
12 let mes="";
13 switch(fecha.getMonth()){
14     case 0: mes="Enero";break;
15     case 1: mes="Febrero";break;
16     case 2: mes="Marzo";break;
17     case 3: mes="Abril";break;
18     case 4: mes="Mayo";break;
19     case 5: mes="Junio";break;
20     case 6: mes="Julio";break;
21     case 7: mes="Agosto";break;
22     case 8: mes="Setiembre";break;
23     case 9: mes="Octubre";break;
24     case 10: mes="Noviembre";break;
25     default: mes="Diciembre"; break;
26 }
27

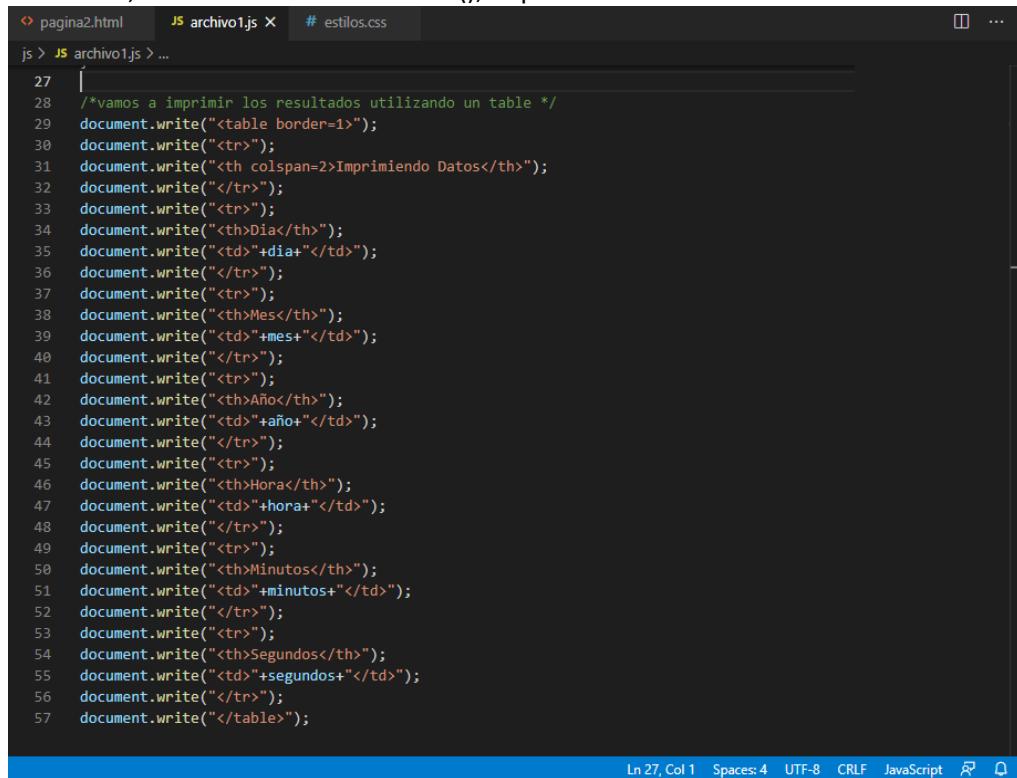
```

Ln 27, Col 1 Spaces: 4 UTF-8 CRLF JavaScript  

Figura 26: Caso práctico

Fuente.- Elaboración Propia

A continuación, utilizando document.write(), imprima los datos en un table



```

pagina2.html    JS archivo1.js # estilos.css
js > JS archivo1.js > ...
27
28 /*vamos a imprimir los resultados utilizando un table */
29 document.write("<table border=1>");
30 document.write("<tr>");
31 document.write("<th colspan=2>Imprimiendo Datos</th>");
32 document.write("</tr>");
33 document.write("<tr>");
34 document.write("<th>Dia</th>");
35 document.write("<td>" + dia + "</td>");
36 document.write("</tr>");
37 document.write("<tr>");
38 document.write("<th>Mes</th>");
39 document.write("<td>" + mes + "</td>");
40 document.write("</tr>");
41 document.write("<tr>");
42 document.write("<th>Año</th>");
43 document.write("<td>" + año + "</td>");
44 document.write("</tr>");
45 document.write("<tr>");
46 document.write("<th>Hora</th>");
47 document.write("<td>" + hora + "</td>");
48 document.write("</tr>");
49 document.write("<tr>");
50 document.write("<th>Minutos</th>");
51 document.write("<td>" + minutos + "</td>");
52 document.write("</tr>");
53 document.write("<tr>");
54 document.write("<th>Segundos</th>");
55 document.write("<td>" + segundos + "</td>");
56 document.write("</tr>");
57 document.write("</table>");

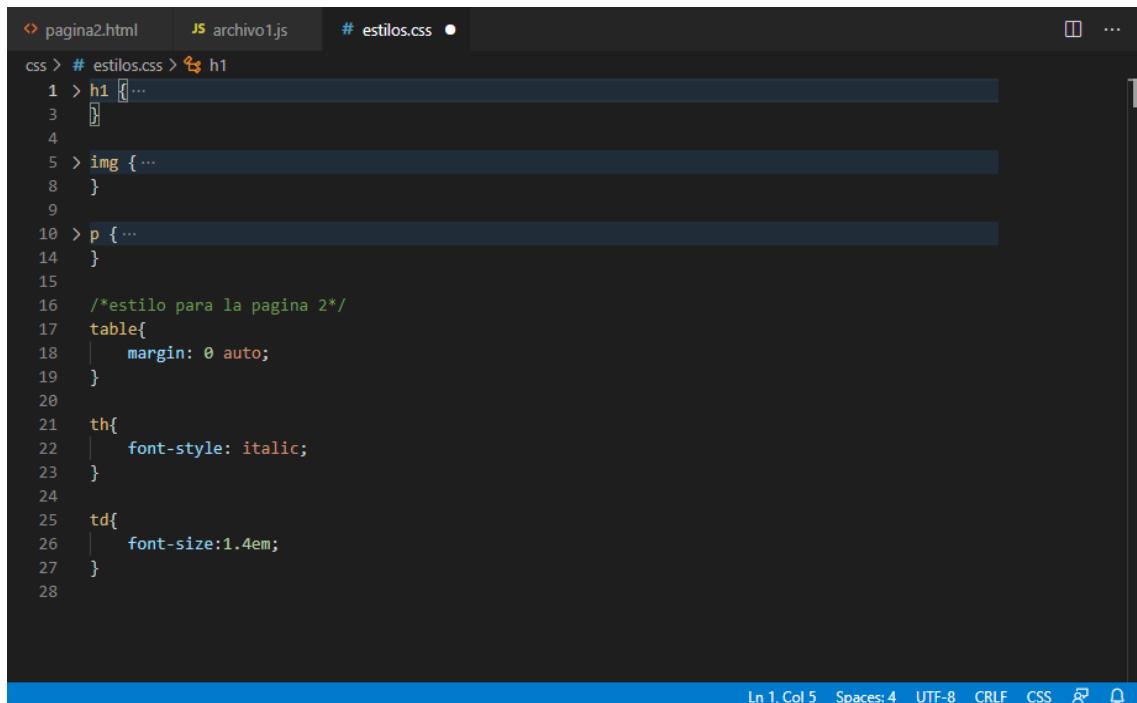

```

Ln 27, Col 1 Spaces: 4 UTF-8 CRLF JavaScript  

Figura 27: Caso práctico

Fuente.- Elaboración Propia

En el archivo estilos.css, defina las reglas a los selectores table, th (cabecera) y td (celda), tal como se muestra.



```

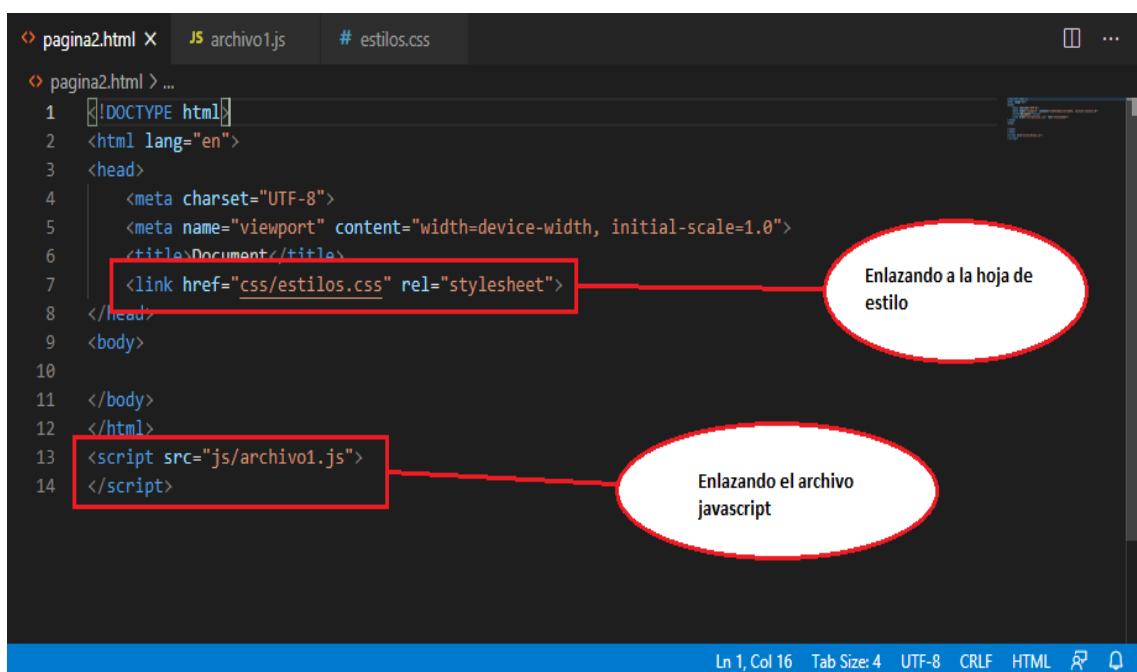
css > # estilos.css > h1
1 > h1 { ...
2   }
3   }
4   }
5 > img { ...
6   }
7   }
8   }
9   }
10 > p { ...
11   }
12   }
13   }
14   }
15   }
16 /*estilo para la pagina 2*/
17 table{
18   margin: 0 auto;
19   }
20   }
21 th{
22   font-style: italic;
23   }
24   }
25 td{
26   font-size:1.4em;
27   }
28

```

Ln 1, Col 5 Spaces: 4 UTF-8 CRLF CSS ⚙️ 🌐

Figura 28: Caso práctico  
Fuente.- Elaboración Propia

En la pagina2.html, enlazamos el archivo de estilos (utilice la etiqueta <link>) y enlazamos el archivo JavaScript (utilice la etiqueta <script>), tal como se muestra.



```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link href="css/estilos.css" rel="stylesheet">
</head>
<body>
</body>
</html>
<script src="js/archivo1.js">
</script>

```

Ln 1, Col 16 Tab Size: 4 UTF-8 CRLF HTML ⚙️ 🌐

Figura 29: Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar la página, ejecuta la tecla F5, selecciona la opción Chrome, visualizamos la página con los datos de la fecha y hora, tal como se muestra.

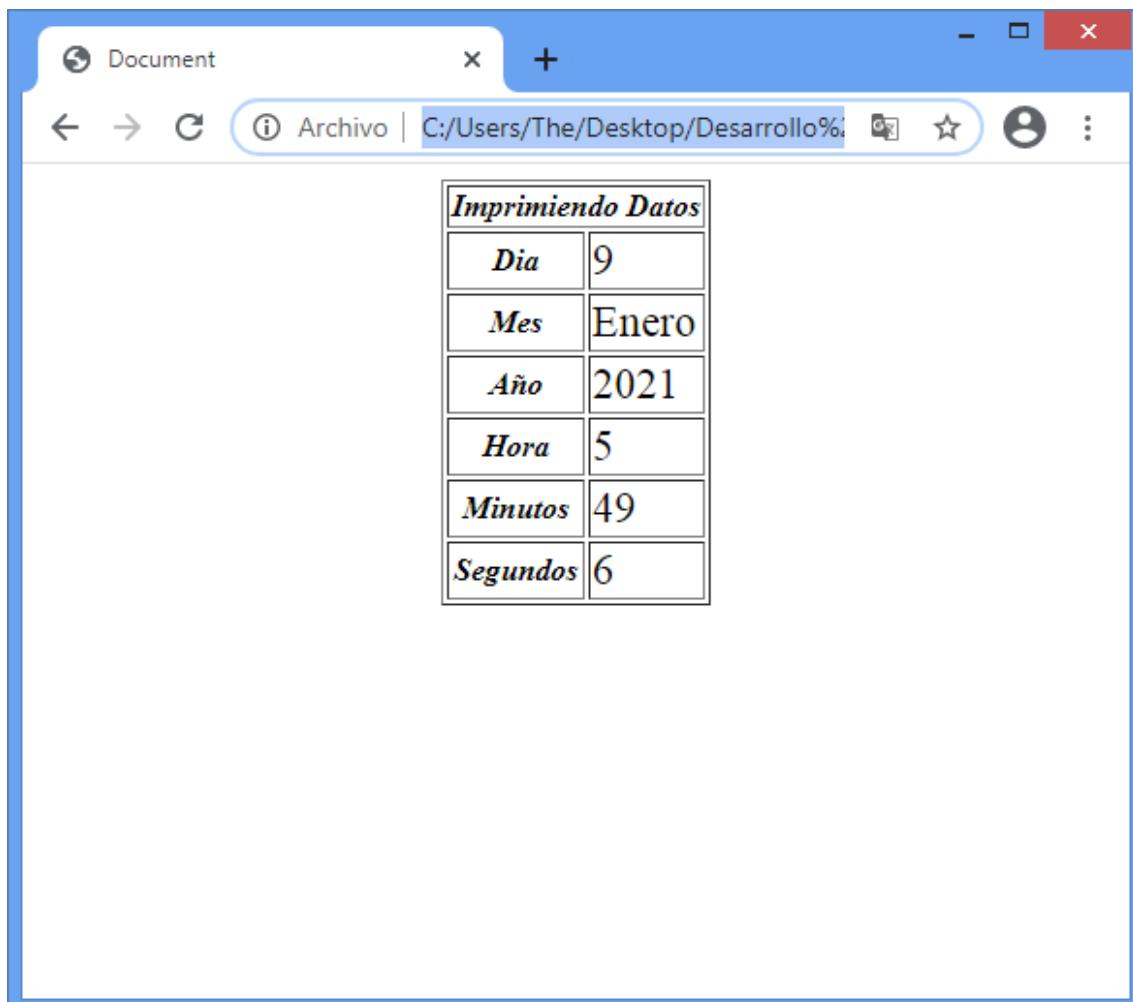


Figura 30: Caso práctico  
Fuente.- Elaboración Propia

## LABORATORIO 3

### Programación básica de JavaScript

Se pide diseñar una página web donde , en un <table> visualice:

1. Imprima los datos del navegador
2. Imprima el ancho y alto de la pagina

Para iniciar con este laboratorio, agregamos una página HTML (Ctrl + N) y al guardar (Ctrl + S), se llamará pagina3.HTML, ubicada en el carpeta raíz del sitio web, tal como se muestra.

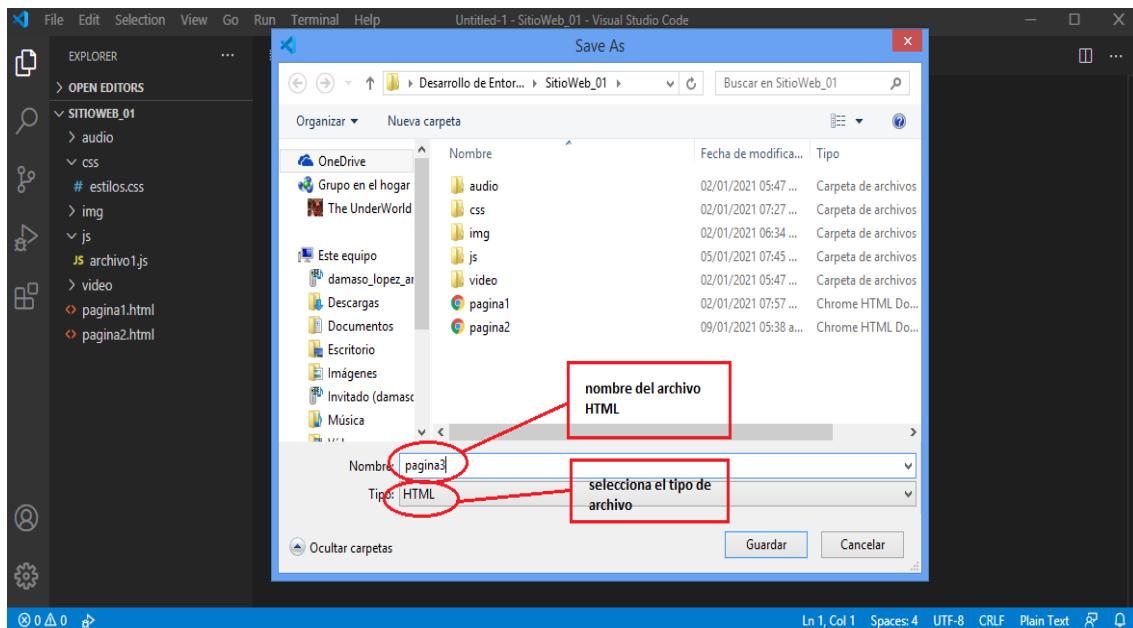


Figura 31: Caso práctico  
Fuente.- Elaboración Propia

Utilizando la abreviatura Emmet, definimos la estructura de la página HTML, selecciona la opción html:5 y presiona ENTER.

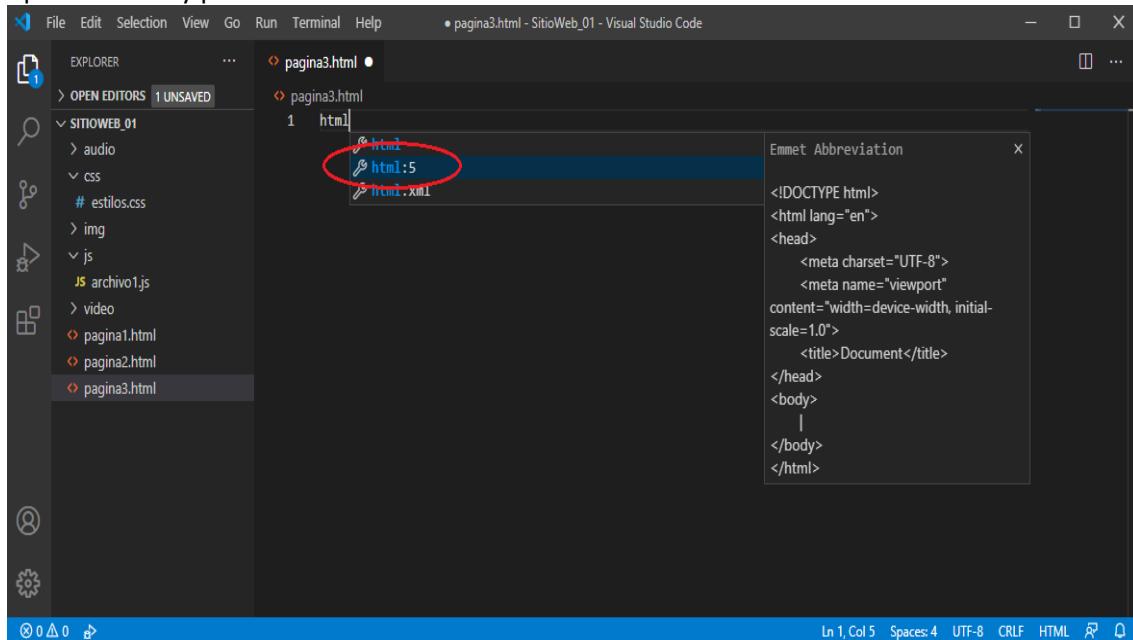
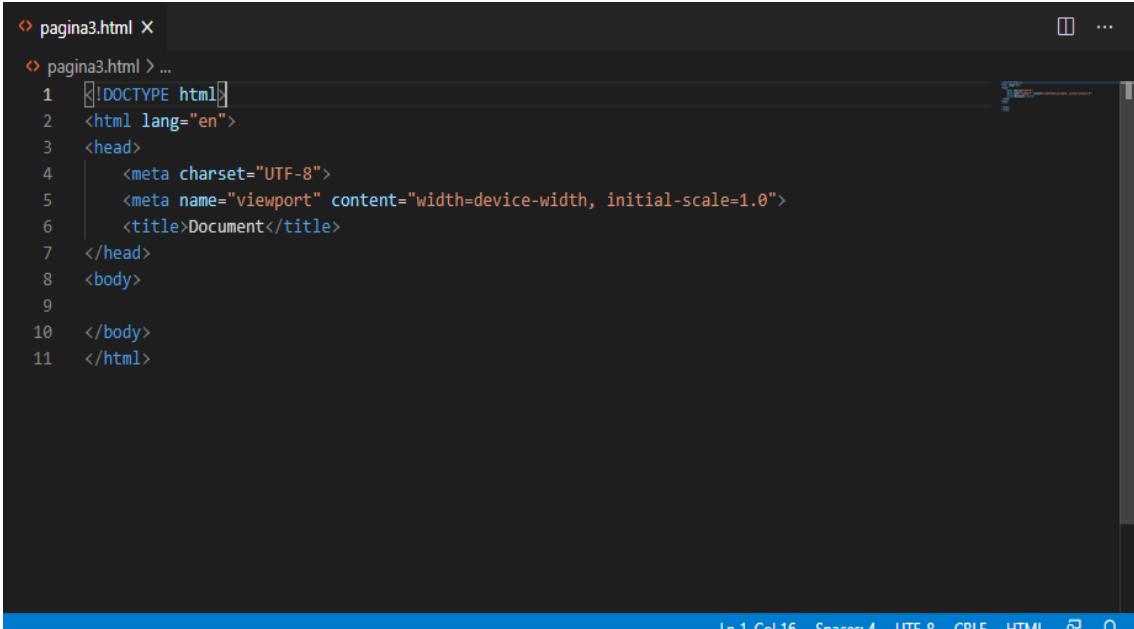


Figura 32: Caso práctico  
Fuente.- Elaboración Propia

A continuación, visualizamos la estructura de la página.



```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
</body>
</html>

```

Figura 33: Caso práctico  
Fuente.- Elaboración Propia

Agregada la pagina3.html, a continuación, agregamos el archivo JavaScript: Ctrl + N (nuevo archivo) y Ctrl + S (guardar), donde selecciona la carpeta JS, asigne el nombre y tipo de archivo, tal como se muestra.

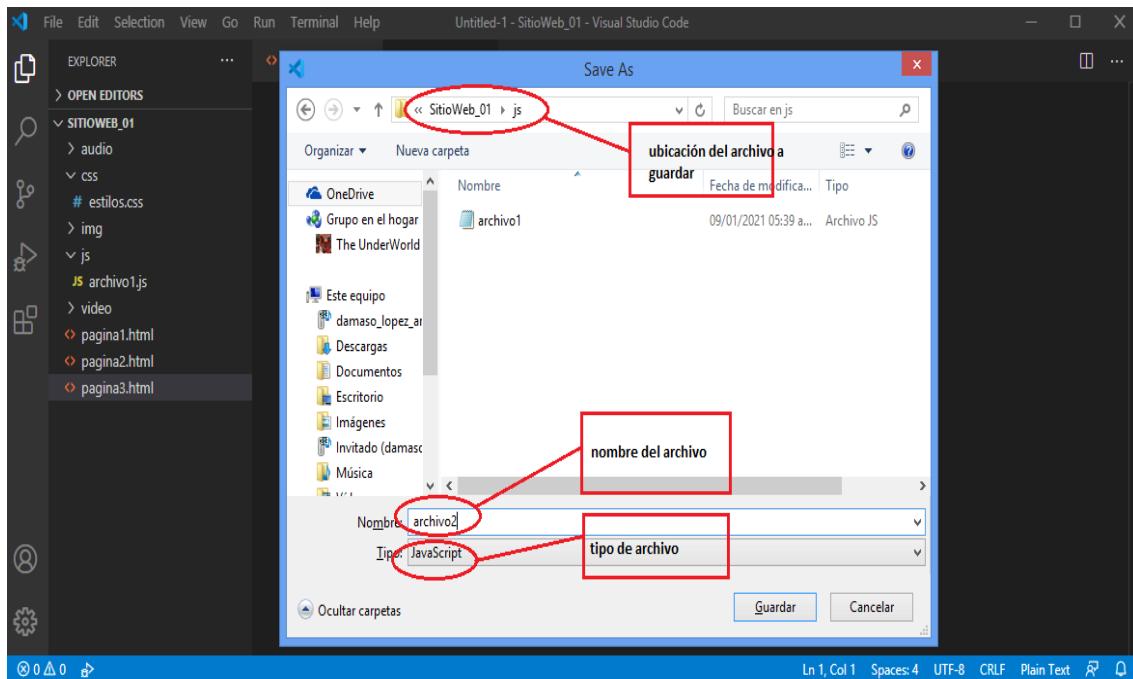
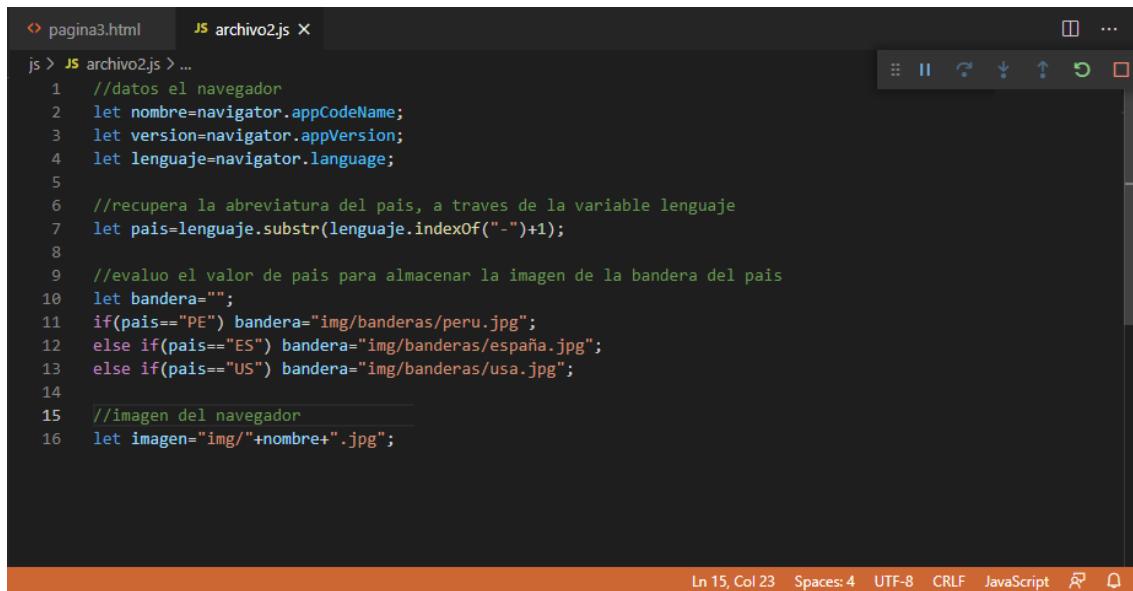


Figura 34: Caso práctico  
Fuente.- Elaboración Propia

En el archivo de JavaScript, defina las variables para almacenar los datos del navegador; además definimos variables para almacenar la imagen del navegador y la imagen de la bandera del país donde está instalado el navegador.



```

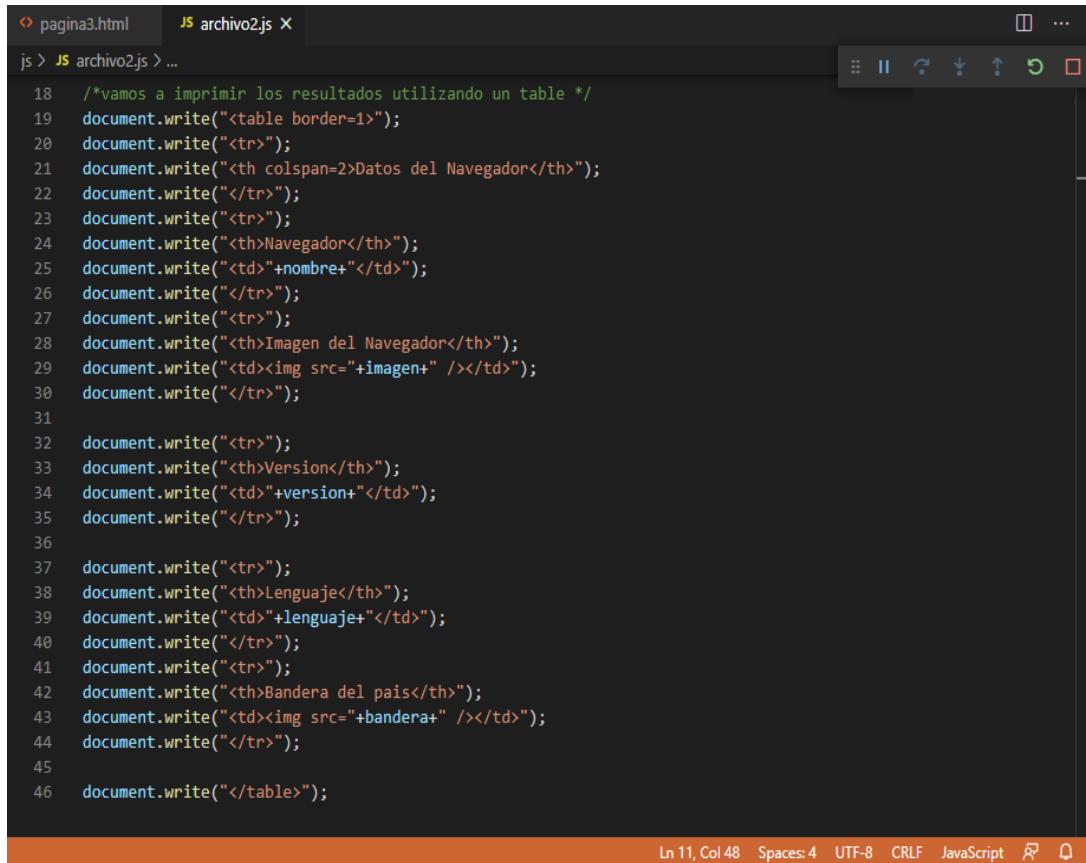
pagina3.html JS archivo2.js
js > JS archivo2.js > ...
1 //datos el navegador
2 let nombre=navigator.appCodeName;
3 let version=navigator.appVersion;
4 let lenguaje=navigator.language;
5
6 //recupera la abreviatura del pais, a traves de la variable lenguaje
7 let pais=lenguaje.substr(lenguaje.indexOf("-")+1);
8
9 //evaluo el valor de pais para almacenar la imagen de la bandera del pais
10 let bandera="";
11 if(pais=="PE") bandera="img/banderas/peru.jpg";
12 else if(pais=="ES") bandera="img/banderas/españa.jpg";
13 else if(pais=="US") bandera="img/banderas/usa.jpg";
14
15 //imagen del navegador
16 let imagen="img/"+nombre+".jpg";

```

Ln 15, Col 23 Spaces: 4 UTF-8 CRLF JavaScript ⚙️ 🗑️

Figura 35: Caso práctico  
Fuente.- Elaboración Propia

A continuación, utilizando document.write(), imprima los datos en un table, tal como se muestra.



```

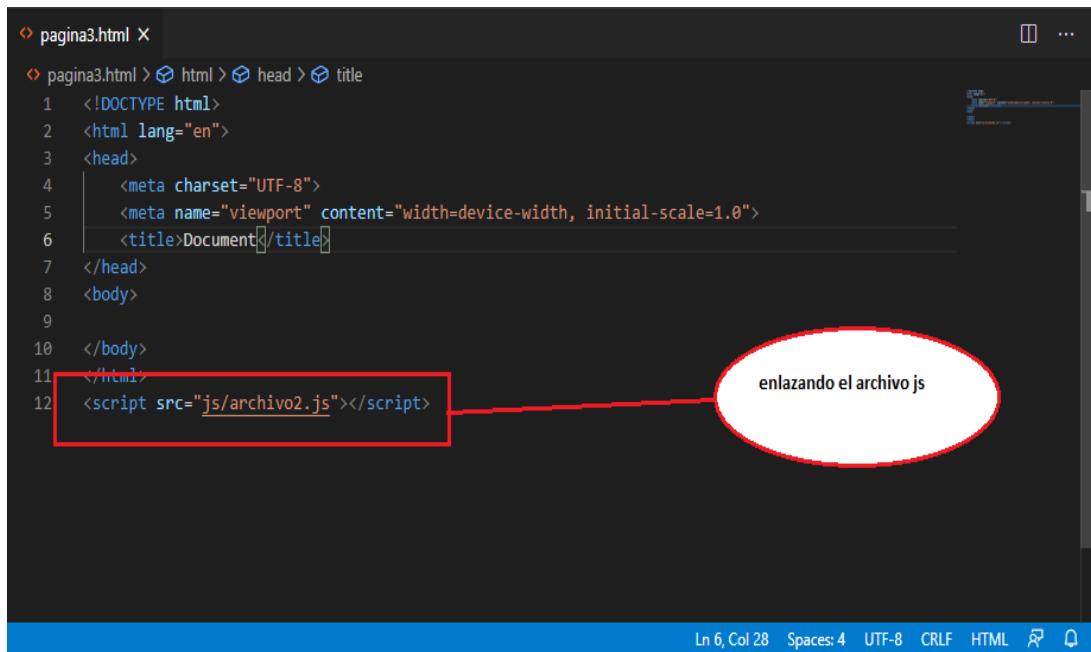
pagina3.html JS archivo2.js
js > JS archivo2.js > ...
18 /*vamos a imprimir los resultados utilizando un table */
19 document.write("<table border=1>");
20 document.write("<tr>");
21 document.write("<th colspan=2>Datos del Navegador</th>");
22 document.write("</tr>");
23 document.write("<tr>");
24 document.write("<th>Navegador</th>");
25 document.write("<td>" + nombre + "</td>");
26 document.write("</tr>");
27 document.write("<tr>");
28 document.write("<th>Imagen del Navegador</th>");
29 document.write("<td></td>");
30 document.write("</tr>");
31
32 document.write("<tr>");
33 document.write("<th>Version</th>");
34 document.write("<td>" + version + "</td>");
35 document.write("</tr>");
36
37 document.write("<tr>");
38 document.write("<th>Lenguaje</th>");
39 document.write("<td>" + lenguaje + "</td>");
40 document.write("</tr>");
41 document.write("<tr>");
42 document.write("<th>Bandera del pais</th>");
43 document.write("<td></td>");
44 document.write("</tr>");
45
46 document.write("</table>");


```

Ln 11, Col 48 Spaces: 4 UTF-8 CRLF JavaScript ⚙️ 🗑️

Figura 36: Caso práctico  
Fuente.- Elaboración Propia

En la pagina3.html, enlazamos el archivo JavaScript (utilice la etiqueta <script>), tal como se muestra.



```
<!-- pagina3.html -->
<!-- pagina3.html > html > head > title
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9
10 </body>
11 </html>
12 <script src="js/archivo2.js"></script>
```

enlazando el archivo js

Figura 37: Caso práctico

Fuente.- Elaboración Propia

Ejecuta página presionando la tecla F5 y selecciona Chrome (preview)



Figura 38: Caso práctico

Fuente.- Elaboración Propia

# Resumen

1. JavaScript es un lenguaje interpretado orientado a las páginas web, con una sintaxis semejante a la del lenguaje Java.
2. Las instrucciones JavaScript para solicitar datos son prompt y el confirm. Las instrucciones para mostrar datos son alert y document.write.
3. Si el cuerpo del if o el cuerpo del else incluyen más de una acción, éstas deben ir encerradas entre llaves de bloque {}.
4. Colocar un ; al final de la condición de un if hace que la acción del if sea nula.
5. Si un case no tiene break, sucederá que al ejecutar las acciones de dicho case se ejecutarán, también, las acciones de los case que siguen hasta encontrar un break o hasta llegar al final del switch.
6. Se puede usar la estructura switch en una toma de decisiones únicamente si las condiciones consisten en comparaciones de una misma variable con una lista de constantes enteras o de carácter.
7. La estructura while es una estructura de propósito general que puede ser usada para resolver cualquier tipo de problema que involucre procesos repetitivos.
8. Si la condición del while resulta falsa la primera vez que se evalúa su condición de control, el while no efectuará ninguna iteración.
9. Si la condición del while no se hace falsa nunca, se genera un bucle infinito.
10. La estructura for es ideal para bucles en los que se conoce el número de iteraciones.
11. Si la condición del for resulta falsa la primera vez que se evalúa su condición de control, el for no efectuará ninguna iteración.
12. La estructura do-while evalúa su condición de control luego de ejecutar su cuerpo de acciones, a diferencia de la estructura while que, primero, prueba su condición de control y, luego, ejecuta su cuerpo de acciones.
13. El cuerpo de acciones de la estructura do-while se ejecutará por lo menos una vez, a diferencia de la estructura while que podría no ejecutar su cuerpo de acciones.

# Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [https://msdn.microsoft.com/es-es/library/6974wx4d\(v=vs.94\).aspx](https://msdn.microsoft.com/es-es/library/6974wx4d(v=vs.94).aspx)
- <http://www.desarrolloweb.com/JavaScript/>
- <http://www.desarrolloweb.com/articulos/tipos-datos-variables-entrada-salida.html>
- <http://www.desarrolloweb.com/manuales/20/>
- [http://librosweb.es/libro/JavaScript/capitulo\\_3/estructuras\\_de\\_control\\_de\\_flujo.html](http://librosweb.es/libro/JavaScript/capitulo_3/estructuras_de_control_de_flujo.html)
- <http://www.arkitzgarro.com/JavaScript/capitulo-5.html>
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Bucles\\_e\\_iteraci%C3%B3n](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Bucles_e_iteraci%C3%B3n)
- <http://www.desarrolloweb.com/articulos/567.php>



# EL MODELO DE OBJETO DE DOCUMENTO

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno diseña y construye páginas para un sitio web aplicando el modelo de objetos de documento DOM.

## TEMARIO

### 2.1 Tema 2 : Modelo DOM

- 2.1.1 : Definición
- 2.1.2 : Árbol de nodos
- 2.1.3 : Tipos de nodos
- 2.1.4 : Acceso directo a los nodos: getelementbyid, getelementsbytagname, getelementsbyclassname, queryselector
- 2.1.5 : Manejo de estilos de los elementos
- 2.1.6 : Acceso directo a los atributos: getattribute, setattribute

## ACTIVIDADES PROPUESTAS

- Mostrar número de enlaces de la página.
- Mostrar la dirección url a la que enlaza el penúltimo enlace.
- Mostrar número de enlaces que enlazan a <http://prueba>.
- Mostrar número de enlaces del tercer párrafo.
- Mostrar u ocultar texto.
- Agregar elementos a una lista.

## 2.1. MODELO DOM

### 2.1.1. Definición

Acorde al W3C el Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API) para documentos validos HTML y bien construidos XML. Define la estructura lógica de los documentos y el modo en que se accede y manipula.

El Modelo de Objetos del Documento (DOM) permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos que un programa JavaScript puede actuar sobre ellos.

Una de las tareas habituales en la programación de aplicaciones web con JavaScript consiste en la manipulación de las páginas web. De esta forma, es habitual obtener el valor almacenado por algunos elementos (por ejemplo, los elementos de un formulario), crear un elemento (párrafos <div>, etc.) de forma dinámica y añadirlo a la página, aplicar una animación a un elemento (que aparezca/desaparezca, que se desplace, etc.).

Todas estas tareas habituales son muy sencillas de realizar gracias a DOM. Sin embargo, para poder utilizar las utilidades de DOM, es necesario "*transformar*" la página original. Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.

Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y, por tanto, la forma en la que se manipulan las páginas.

### 2.1.2. Árbol de nodos

DOM transforma todos los documentos HTML en un conjunto de elementos llamados *nodos*, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama "árbol de nodos".

La siguiente página HTML sencilla:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Página sencilla</title>
</head>

<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Se transforma en el siguiente árbol de nodos:

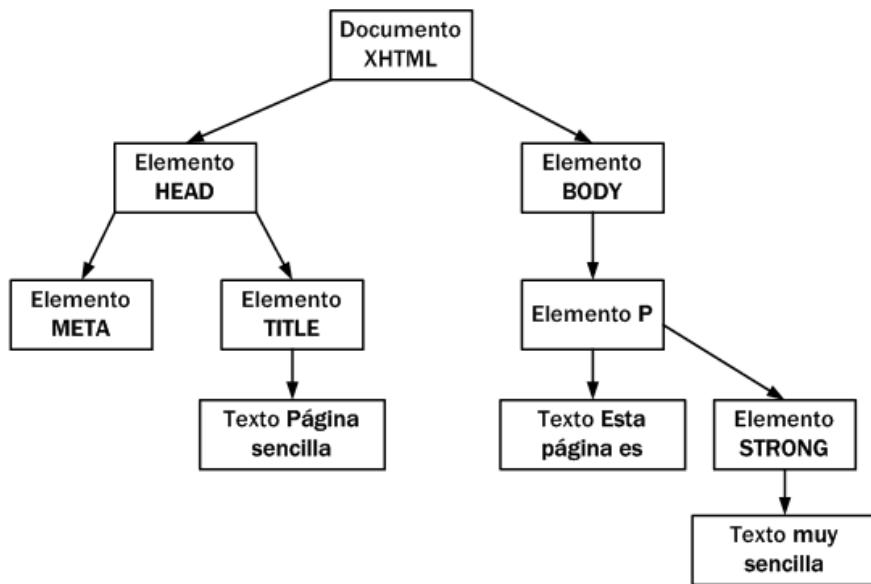


Figura 39: Modelo DOM  
Fuente.- Elaboración Propia

En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.

La raíz del árbol de nodos de cualquier página HTML siempre es la misma: un nodo de tipo especial denominado "*Documento*".

A partir de ese nodo raíz, cada etiqueta HTML se transforma en un nodo de tipo "*Elemento*". La conversión de etiquetas en nodos se realiza de forma jerárquica. De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY. A partir de esta derivación inicial, cada etiqueta HTML se transforma en un nodo que deriva del nodo correspondiente a su "*etiqueta padre*".

La transformación de las etiquetas HTML habituales genera dos nodos: el primero es el nodo de tipo "*Elemento*" (correspondiente a la propia etiqueta HTML) y el segundo es un nodo de tipo "*Texto*" que contiene el texto encerrado por esa etiqueta HTML.

Así, la siguiente etiqueta HTML:

```
<title>Página sencilla</title>
```

Genera los siguientes dos nodos:



Figura 40: Modelo DOM  
Fuente.- Elaboración Propia

De la misma forma, la siguiente etiqueta HTML:

```
<p>Esta página es <strong>muy sencilla</strong></p>
```

Genera los siguientes nodos:

- Nodo de tipo "*Elemento*" correspondiente a la etiqueta `<p>`.
- Nodo de tipo "*Texto*" con el contenido textual de la etiqueta `<p>`.
- Como el contenido de `<p>` incluye en su interior otra etiqueta HTML, la etiqueta interior se transforma en un nodo de tipo "*Elemento*" que representa la etiqueta `<strong>` y que deriva del nodo anterior.
- El contenido de la etiqueta `<strong>` genera a su vez otro nodo de tipo "*Texto*" que deriva del nodo generado por `<strong>`.

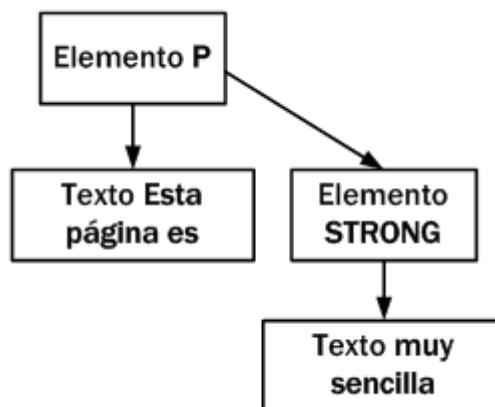


Figura 41: Modelo DOM  
Fuente.- Elaboración Propia

La transformación automática de la página en un árbol de nodos siempre sigue las mismas reglas:

- Las etiquetas HTML se transforman en dos nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primero y consiste en el contenido textual de la etiqueta.
- Si una etiqueta HTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

Como se puede suponer, las páginas HTML habituales producen árboles con miles de nodos. Aun así, el proceso de transformación es rápido y automático, siendo las funciones proporcionadas por DOM las únicas que permiten acceder a cualquier nodo de la página de forma sencilla e inmediata.

Decimos que una página web es un documento HTML. Este documento puede ser representado de diferentes maneras:

- Representación web:** como una página web en un navegador donde vemos imágenes, texto, colores, etc.
- Representación texto:** como un texto plano (código HTML) que podemos visualizar en cualquier editor de textos como el bloc de notas de Windows o Notepad++ o cualquier otro.

- c) **Representación DOM:** como un árbol donde los elementos de la página web están organizados jerárquicamente, con nodos superiores (nodos padre o parent) y nodos que derivan de los nodos padre (nodos hijo o child).

El DOM no es parte de JavaScript, de hecho, puede ser utilizado por otros lenguajes de programación. No obstante, el DOM está íntimamente ligado a JavaScript ya que JavaScript lo utilizará con profusión para acceder y modificar las páginas web dinámicamente.

Decimos que conforme al DOM la página web se representa como un árbol de nodos, interconectados y relacionados de acuerdo con una jerarquía.

El DOM permite un acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto. Para comprender más fácilmente véase el siguiente ejemplo:

#### **Siempre use el DOCTYPE correcto**

El DOCTYPE (abreviado del inglés “document type declaration”, declaración del tipo de documento) informa cual versión de (X)HTML se usará para validar el documento; existen varios tipos a seleccionar. El DOCTYPE, debe aparecer siempre en la parte superior de cada página HTML y siendo un componente clave de las páginas web “obedientes” a los estándares.

En caso de usarse un DOCTYPE incompleto, no actualizado o simplemente no usarlo llevará al navegador a entrar en modo raro o extraño, donde el navegador asume que se ha programado fuera de los estándares.

Todavía todos los navegadores actuales no son capaces de procesar correctamente todos los tipos de documentos, sin embargo, muchos de ellos funcionan correctamente en los navegadores más utilizados actualmente, tales como:

HTML 4.01 Strict y Transitional, XHTML 1.0 Strict y Transitional los que se comportan del modo correcto en Internet Explorer (versión 6, 7 Beta), Mozilla y Opera 7. De ahora en adelante se adoptará para cada ejemplo HTML 4.01 Strict :

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

Resultando una única línea de código, o dos líneas con un salto de línea después de EN”.

#### **2.1.3. Tipos de nodos**

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

1. **Document**, nodo raíz del que derivan todos los demás nodos del árbol.
2. **Element**, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
3. **Attr**, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par atributo=valor.
4. **Text**, nodo que contiene el texto encerrado por una etiqueta XHTML.
5. **Comment**, representa los comentarios incluidos en la página XHTML.

Los otros tipos de nodos existentes que no se van a considerar son `DocumentType`, `CDataSection`, `DocumentFragment`, `Entity`, `EntityReference`, `ProcessingInstruction` y `Notation`.

#### **2.1.4 Acceso directo a los nodos: `getelementbyid`, `getelementsbytagname`, `getelementsbyclassname`, `queryselector`**

El DOM permite un acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto. Para comprender más fácilmente véase el siguiente ejemplo:

```
<body>
<p>Esto es un párrafo que contiene <a href="#">un enlace</a> en el medio. </p>
<ul>
<li>Primer punto en la lista</li>
<li>Otro punto en la lista</li>
</ul>
</body>
```

Como puede verse un elemento [a] se encuentra localizado dentro de un elemento [p] del HTML, convirtiéndose en un nodo hijo, o simplemente hijo del nodo [p], de manera similar [p] es el nodo padre. Los dos nodos li son hijos del mismo padre, llamándose nodos hermanos o simplemente hermanos.

Es importante comprender la diferencia entre elementos y nodos de textos. Los elementos comúnmente son asociados a las etiquetas. En HTML todas las etiquetas son elementos, tales como `<p>`, `<img>` y `<div>` por lo que tienen atributos y contienen nodos hijos. Sin embargo, los nodos de textos no poseen atributos e hijos.

Afortunadamente, JavaScript permite acceder a cada uno de los elementos de una página utilizando tan sólo algunos métodos y propiedades.

Método	Descripción
<code>getElementById</code>	Método que devuelve el elemento de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función
<code>getElementsByTagName</code>	Método que devuelve todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.
<code>getElementsByClassName</code>	Método que devuelve todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función
<code>querySelector</code>	Método que devuelve el primer elemento del documento (utilizando un recorrido primero en profundidad preordenado de los nodos del documento) que coincide con el grupo especificado de selectores.
<code>querySelectorAll</code>	El método que <code>SelecterAll()</code> , devuelve una NodeList estática (no viva) que representa una lista de

	elementos del documento que coinciden con el grupo de selectores indicados.
--	---

### Función getElementById()

La función `getElementById()` es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de una función que permite acceder directamente a un nodo y poder leer o modificar sus propiedades.

La función `getElementById()` devuelve el elemento XHTML cuyo atributo `id` coincide con el parámetro indicado en la función. Como el atributo `id` debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

```
var cabecera = document.getElementById("cabecera");

<div id="cabecera">
  <a href="/" id="logo">...</a>
</div>
```

### Función getElementsByTagName()

La función `getElementsByTagName(nombreEtiqueta)` obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

El siguiente ejemplo muestra cómo obtener todos los párrafos de una página XHTML:

```
var parrafos = document.getElementsByTagName("p");
```

El valor que se indica delante del nombre de la función (en este caso, `document`) es el nodo a partir del cual se realiza la búsqueda de los elementos. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor `document` como punto de partida de la búsqueda.

El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales.

### Función getElementsByClassName()

La función `getElementsByName()` es similar a la anterior, pero en este caso se buscan los elementos cuyo selector class sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

```
var parrafoEspecial = document.getElementsByClassName("especial");

<p class="prueba">...</p>
<p class="especial">...</p>
<p>...</p>
```

### Función querySelector()

Devuelve el primer elemento del documento que coincide con el grupo especificado de selectores.

Sintaxis:

```
element = document.querySelector(selectores);
```

Esta función devuelve null si no se encuentran coincidencias, de lo contrario, retorna el primer elemento encontrado. Si el selector coincide con un ID y este ID es usado erróneamente varias veces en el documento, devuelve el primer elemento encontrado.

La cadena de caracteres que se pasa como argumento a querySelector debe seguir la sintaxis CSS.

### Función querySelectorAll()

El método querySelectorAll() de un Element devuelve una **NodeList** estática (no viva) que representa una lista de elementos del documento que coinciden con el grupo de selectores indicados.

Sintaxis:

```
elementList = parentNode.querySelectorAll(selectors);
```

Una vez que se devuelve la **NodeList** de los elementos que coinciden, se puede examinar como cualquier array. Si el array está vacío (lo que significa que su propiedad length es 0), entonces es que no se encontraron coincidencias.

En cualquier caso, se puede simplemente utilizar la notación estándar de los arrays para acceder al contenido de la lista. Se puede usar cualquier sentencia de iteración, como, por ejemplo:

```
var items = userList.querySelectorAll('.item');

items.forEach(function(userItem) {
  deleteUser(userItem);
});
```

## 2.1.5. Manejo los estilos de los elementos

Como se ha visto, los atributos que le son asignados a las etiquetas HTML están disponibles como propiedades de sus correspondientes nodos en el DOM. Las propiedades de estilo pueden ser aplicadas a través del DOM.

Cada atributo CSS posee una propiedad del DOM equivalente, formándose con el mismo nombre del atributo CSS, pero sin los guiones y llevando la primera letra de las palabras a mayúsculas. Véase el siguiente ejemplo para mayor entendimiento donde se utiliza un atributo CSS modelo:

algun-atributo-css

Tendrá como equivalente la siguiente propiedad o método en JavaScript:

algunAtributoCss

Por tanto, para cambiar el atributo CSS font-family de un elemento, podría realizarse de lo siguiente:

```
ancla.style.fontFamily = 'sans-serif';
```

Los valores CSS en JavaScript serán en su mayoría del tipo cadena; por ejemplo: font-size, pues posee dimensiones tales como "px", "%". Sólo los atributos completamente numéricos, tales como z-index serán del tipo entero. En muchos casos es necesario aparecer y desaparecer un determinado elemento, para ellos se utiliza el atributo CSS display, por ejemplo, para desaparecer:

```
ancla.style.display = 'none';
```

Luego para volverlo a mostrar se le asigna otro valor:

```
ancla.style.display = 'inline';
```

El siguiente código permite aplicar estilo de forma general al documento y el segundo es más específico, al ser dirigido a un bloque o elemento definido por un identificador ID.

```
document.body.style.fontSize="24px"  
document.getElementById("id").style.property="value"  
var el = document.getElementById(id);  
el.style.color = "red";  
el.style.fontSize = "15px";  
el.style.backgroundColor = "#FFFFFF";
```

Lista de propiedades de JavaScript para modificar el estilo de las páginas

Propiedad	Descripción
background	Establece todas las propiedades del background o fondo en una única declaración de forma abreviada, son 5 las propiedades separadas por un espacio: background-color, background-image, background-repeat, background-attachment, background-position Se usa de la siguiente forma: Object.style.background="color image repeat attachment position" Por ejemplo: document.body.style.background="#f3f3f3 url('foto.png') no-repeat right top";
backgroundAttachment	Especifica cuando la imagen empleada como fondo permanece fija o se desplaza. Las opciones son: scroll Se desplaza con la página (Predeterminado) fixed Permanece fija
backgroundColor	Especifica el color del fondo
backgroundImage	Especifica la ubicación de la imagen a emplear como fondo. Por ejemplo: document.body.style.backgroundImage="url('foto.png')";
backgroundPosition	Posición de la imagen utilizada. Se usan dos valores, si solo se especifica uno la imagen será centrada. Los valores pueden ser: top left, top center, top right, center left, center, center right, bottom left, bottom center, bottom right x% y% xpos ypos
backgroundRepeat	Especifica si la imagen empleada se repite para llenar toda el área de la página. repeat Es repetida en las dos dimensiones, es el valor predeterminado. repeat-x Solo se repite en el eje horizontal repeat-y Solo se repite en el eje vertical no-repeat No es repetida
border	Establece las propiedades del borde en una sola declaración de la siguiente forma: Object.style.border="width style color" Por ejemplo: document.getElementById("ejemplo").style.border="thick solid green"; Las restantes propiedades que se pueden usar de forma individual para definir el estilo del borde son: borderBottom borderBottomColor

	borderBottomStyle borderBottomWidth borderColor borderLeft borderLeftColor borderLeftStyle borderLeftWidth borderRight borderRightColor borderRightStyle borderRightWidth borderStyle borderTop borderTopColor borderTopStyle borderTopWidth borderWidth
outline	<p>Especifica las propiedades de outline (borde de fuente) en una sola declaración de la siguiente forma:</p> <pre>Object.style.outline="width style color"</pre> <p>Por ejemplo:</p> <pre>document.getElementById("ejemplo").style.outline="thick solid #0000FF";</pre> <p>Las restantes propiedades que se pueden usar de forma individual para definir el estilo outline son:</p> <pre>outlineColor outlineStyle outlineWidth</pre>
listStyle	<p>Especifica las siguientes propiedades en una sola declaración:</p> <p>list-style-image, list-style-position y list-style-type, se usa:</p> <pre>Object.style.listStyle="type position image"</pre> <p>También se pueden usar de forma individual:</p> <pre>listStyleImage listStylePosition listStyleType</pre>
margin	<p>Establece todas las propiedades de los márgenes en una sola declaración. Esta propiedad puede establecerse indicando desde 1 a 4 valores.</p> <ul style="list-style-type: none"> <li>• Un valor, por ejemplo: div {margin: 50px} todos los márgenes serán de 50px</li> <li>• Dos valores, por ejemplo: div {margin: 50px 10px} top (superior) y bottom (inferior) serán de 50px, left (izquierda) y right (derecha) serán de 10px.</li> <li>• Tres valores, por ejemplo: div {margin: 50px 10px 20px} el valor de top será 50px, left y right será 10px, bottom será 20px.</li> <li>• Cuatro valores, por ejemplo: div {margin: 50px 10px 20px 30px} el valor de top será 50px, right será de 10px, bottom</li> </ul>

	<p>será de 20px, left será de 30px.</p> <p>Los valores se pueden definir de tres formas alternativas "% length auto"</p> <p>Por ejemplo:</p> <pre>document.getElementById("ejemplo").style.margin="2px 2px 5px 5px";</pre> <p>También se pueden usar de forma individual:</p> <ul style="list-style-type: none"> <li>marginBottom</li> <li>marginLeft</li> <li>marginRight</li> <li>marginTop</li> </ul>
padding	<p>Especifica de forma conjunta los valores del padding (espaciado) de un elemento, se pueden usar hasta cuatro valores. Para emplear los valores utiliza el mismo método de margin. Por ejemplo:</p> <pre>document.getElementById("ejemplo").style.padding="10px 0 5px 0";</pre> <p>También se pueden usar de forma individual:</p> <ul style="list-style-type: none"> <li>paddingBottom</li> <li>paddingLeft</li> <li>paddingRight</li> <li>paddingTop</li> </ul>
cssText	Especifica una declaración de estilo como una cadena de texto.
clear	Especifica la posición de un elemento de forma relativa a un objeto que flota.
clip	Especifica que parte de un elemento posicionado es visible.
cssFloat	<p>Establece la alineación horizontal de un elemento. Pueden usarse tres valores de la siguiente forma:</p> <pre>Object.style.cssFloat="left right none"</pre> <p>Por ejemplo:</p> <pre>document.getElementById("ejemplo").style.cssFloat="left";</pre>
cursor	<p>Establece el estilo a emplearse en el cursor del ratón de la siguiente forma:</p> <pre>Object.style.cursor="valor"</pre> <p>Los valores pueden ser: auto(predeterminado), crosshair, e-resize, help, move, n-resize, ne-resize, nw-resize, pointer, s-resize, se-resize, sw-resize, text, url, w-resize, wait.</p> <p>Por ejemplo:</p> <pre>document.getElementById("ejemplo").style.cursor="pointer";</pre>
display	<p>Define la forma en que se muestra un elemento HTML, los métodos más empleados son: "inline" o "block", también es muy usado para ocultar elementos de la forma siguiente:</p> <pre>document.getElementById("ejemplo").style.display="none";</pre>

overflow	Especifica que hacer si el contenido de un elemento rebasa el espacio que proporciona este.
position	Especifica el tipo de posicionamiento usado para un elemento, pueden usarse los siguientes valores (static, relative, absolute o fixed)
verticalAlign	Especifica la alineación vertical de un elemento
visibility	Especifica la visibilidad de un elemento
zIndex	Especifica el orden del posicionamiento de un elemento
orphans	Especifica el mínimo número de líneas para un elemento que tiene que ser visible en la parte inferior de la página.
widows	Especifica el mínimo número de líneas para un elemento que tiene que ser visible en la parte superior de la página.
borderCollapse	Establece si el borde de una tabla debe colapsar en una simple línea
borderSpacing	Espaciado del borde en una tabla
captionSide	Posición del elemento caption de una tabla
emptyCells	Especifica si se debe mostrar el borde y fondo de una celda vacía
color	Establece el color del texto.
direction	Establece la dirección del texto
font	<p>Establece todas las propiedades del elemento fuente en una sola declaración.          Pueden usarse los valores: font-style, font-variant, font-weight, font-size, line-height y font-family.          Hazlo de la siguiente forma:          Object.style.font="style variant weight size/lineHeight family"          Por ejemplo:          document.getElementById("ejemplo").style.font="italic bold          18px arial,serif";</p> <p>También pueden usarse de forma individual las siguientes propiedades:</p> <ul style="list-style-type: none"> <li>fontFamily</li> <li>fontSize</li> <li>fontSizeAdjust</li> <li>fontStyle</li> <li>fontVariant</li> <li>fontWeight</li> </ul>
letterSpacing	Especifica el espacio entre caracteres en el texto.
lineHeight	Especifica el espacio entre líneas en el texto.

textAlign	Especifica la alineación horizontal del texto.
textDecoration	Especifica el estilo de decoración del texto.
textIndent	Establece la indentación utilizada en la primera línea del texto.
textTransform	Establece la propiedad Transform usada en el texto.
wordSpacing	Establece el espacio entre palabras en el texto.
	Para la posición y tamaño de cada elemento se pueden usar las siguientes propiedades: <i>Bottom, left, right, top</i> <i>Height, width, maxHeight, maxWidth , minHeight ,minWidth</i>

### 2.1.6. Acceso directo a los atributos: `getAttribute`, `setAttribute`

Las partes más frecuentemente usadas de un elemento HTML son sus atributos, tales como: id, class, href., title, estilos CSS (style), entre muchas otras piezas de información que pueden ser incluidas en una etiqueta HTML.

Los atributos de una etiqueta son traducidos por el navegador en propiedades de un objeto. Dos métodos existen para leer y escribir los atributos de un elemento, `getAttribute` permite leer el valor de un atributo mientras que `setAttribute` permite su escritura.

En ocasiones se hace necesario ver las propiedades y métodos de un determinado elemento, esto puede realizarse utilizando la función `getAttribute()` mediante la siguiente función utilitaria:

```
function inspector(el) {
  var str = "";
  for (var i in el){
    str+=i + ":" + el.getAttribute(i) + "\n";
  }
  alert(str);
}
```

Para usar la función `inspector()` tan sólo debe pasarle la referencia al elemento, continuando con el ejemplo anterior resulta:

```
var ancla = document.getElementById("editor");
inspector(ancla);
```

Para modificar el atributo title del hipervínculo, elemento referenciado por la variable `ancla`, se usará el `setAttribute()`, pasándole el nombre del atributo y el valor:

```
var ancla = document.getElementById("editor");
ancla.setAttribute("title", "Artículos de programación");
var nuevoTitulo = ancla.getAttribute("title");
```

El valor de la variable `nuevoTitulo` es ahora “Artículos de programación”.

## Usando innerHTML

En aplicaciones complejas donde es necesario crear varios elementos a la vez, el código JavaScript generado puede ser extenso recurriendo a la propiedad innerHTML. Dicha propiedad fue introducida por Microsoft permitiendo leer y escribir el contenido HTML de un elemento.

Por ejemplo, puede crearse fácilmente una tabla con múltiples celdas e insertarla luego en la página con innerHTML:

```
var tabla = '<table border="0">';
tabla += '<tr><td>Celda 1</td><td>Celda 2</td><td> Celda 3</td></tr>';
tabla += '</table>';
document.getElementById("datos").innerHTML = tabla;
```

## LABORATORIO 1

### Trabajando con el modelo DOM

Implementa una página web la cual permita visualizar la fecha del sistema e ingresar el usuario a través de un prompt donde:

- Si el usuario es “cibertec”, cambiar el color de fondo de la página (skyblue), visualizar el texto ingresado y visualizar la imagen del usuario cibertec,
- Si el usuario no es “cibertec”, cambiar el color a gris y visualizar la imagen del usuario desconocido

### 1. Creando el Sitio Web

Defina una carpeta llamado SitioWeb\_02, en ella crea las siguientes subcarpetas, tal como se muestra.

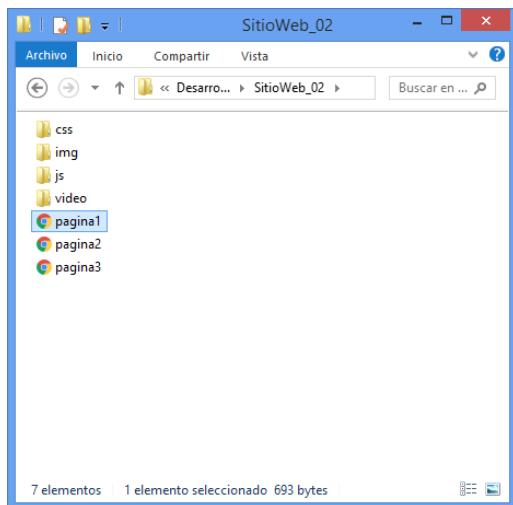


Figura 42: Caso práctico  
Fuente.- Elaboración Propia

### 2. Trabajando con Visual Studio Code

Para crear una página web, abrir el aplicativo Visual Studio Code.

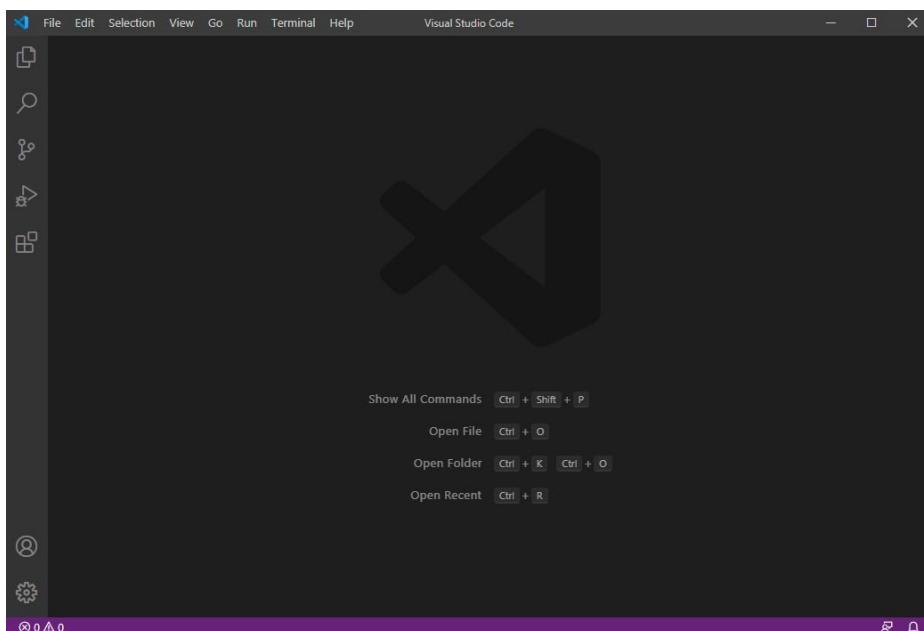


Figura 43 : Caso práctico  
Fuente.- Elaboración Propia

Para trabajar con la carpeta llamada SitioWeb\_02 en Visual Studio Code, arrastre la carpeta hacia la ventana del Visual, tal como se muestra; y luego soltarla.

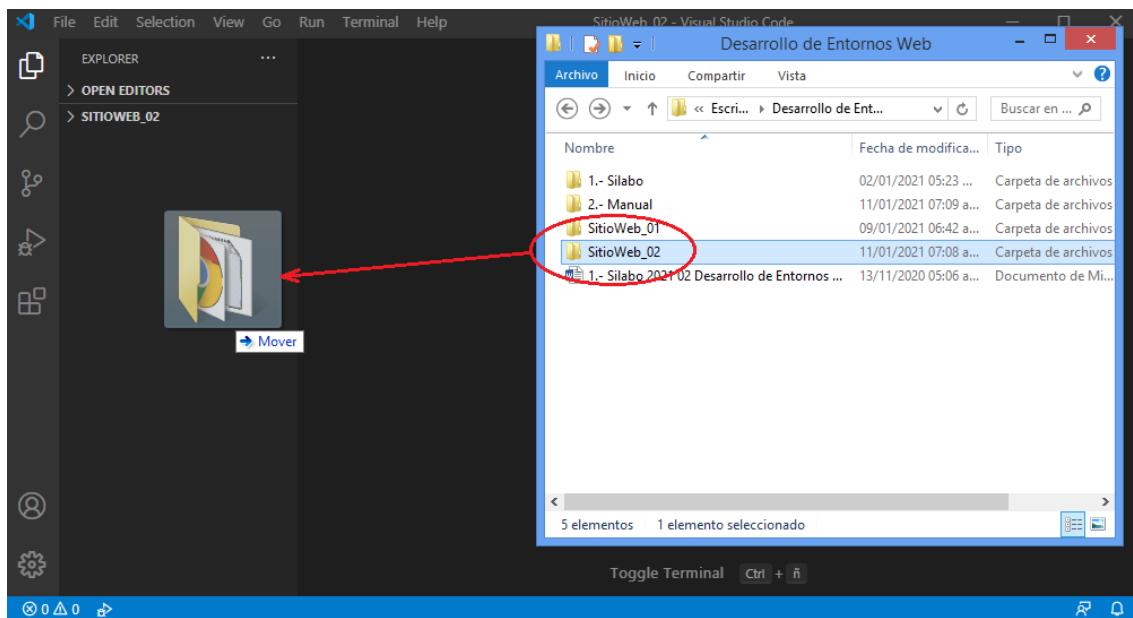


Figura 44 : Caso práctico  
Fuente.- Elaboración Propia

Donde se visualiza la carpeta en la parte izquierda de la ventana del aplicativo

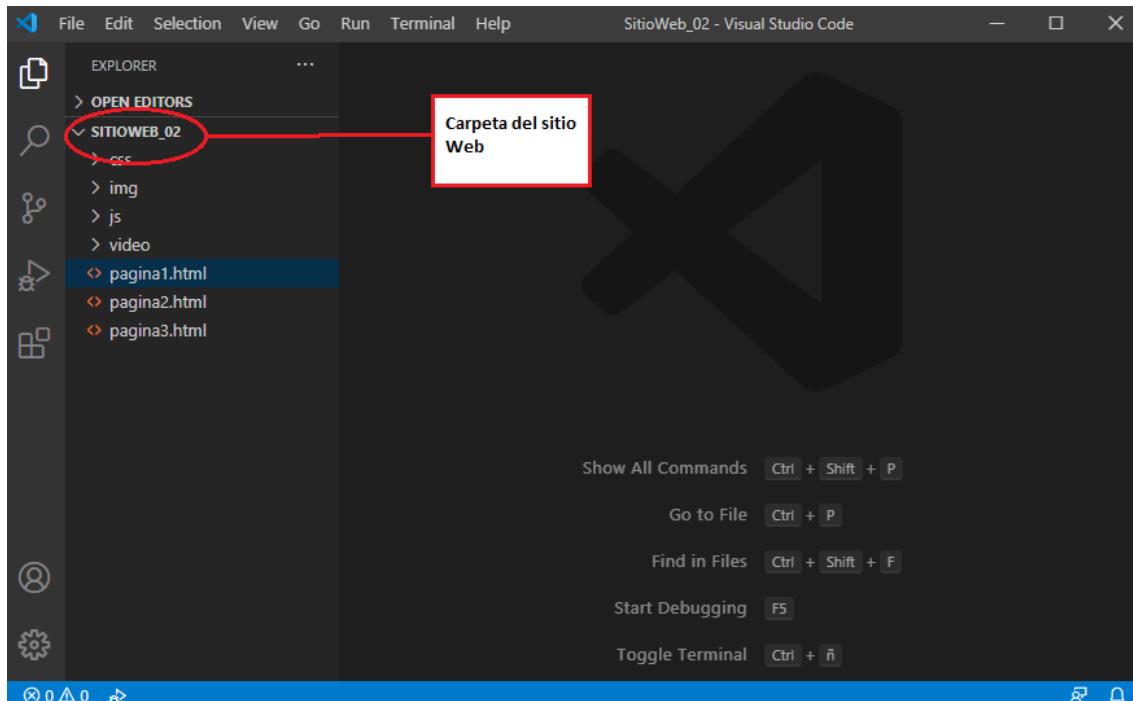


Figura 45: Caso práctico  
Fuente.- Elaboración Propia

### 3. Diseñando la página web

En esta etapa vamos a diseñar la página1.html, tal como se muestra.



Figura 46 : Caso práctico  
Fuente.- Elaboración Propia

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_1.css. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

    <!--pagina1.html -->
    <!--pagina1.html ><!-- html --><!-- body --><!-- div#principal -->
    1   <!DOCTYPE html>
    2   <html>
    3   <head>
    4       <title></title>
    5       <link href="css/estilos_1.css" rel="stylesheet" />
    6   </head>
    7   <body>
    8       <div id="principal">
    9
    10      <header>
    11          <h1>Portal JavaScript</h1>
    12          
    13      </header>
    14
    15      <nav>
    16          <div id="pf">Fecha?</div>
    17          <div id="ph">
    18              <span id="usuario">Usuario?</span>
    19              <img id="imgusuario" src="" />
    20          </div>
    21      </nav>
    22
    23      <section>
    24          <div id="blok1"></div>
    25          <div id="blok2"></div>
    26      </section>
    27
    28  </div>
    29 </body>
    30 </html>
    31

```

Annotations on the code:

- Line 5: "Enlazando a la hoja de estilo: estilos\_1"
- Line 11: "bloque header, agrupando: h1 e img"
- Line 19: "bloque nav, agrupando donde bloques: programacion"
- Line 25: "bloque section"

Figura 47: Caso práctico  
Fuente.- Elaboración Propia

**Definiendo el archivo estilos\_1.css**

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

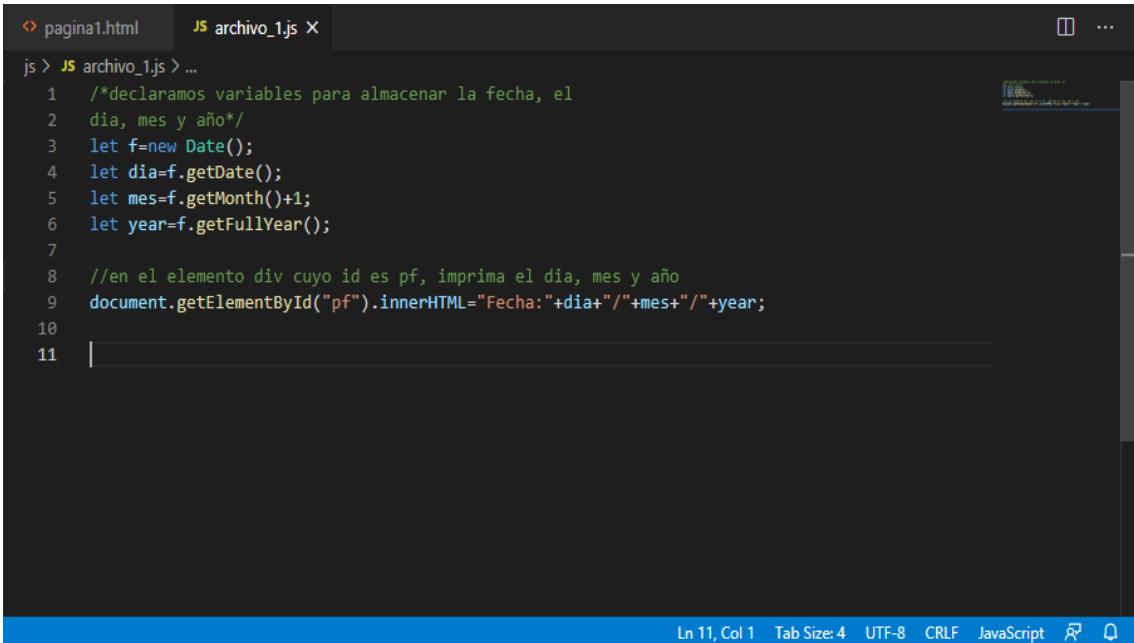
```
body{  
    background-color: gray;  
}  
  
#principal{  
    width: 80%;  
    float: left;  
    border: 5px solid;  
    margin: 0 10% 0 10%;  
    border-radius: 20px;  
    padding-bottom: 10px;  
    padding-top: 10px;  
    background-color: white;  
}  
  
h1{  
    text-align: center;  
    color: gray;  
}  
  
#imgcab{  
    width: 100%;  
    height: 300px;  
    float: left;  
}  
  
#pf, #ph{  
    width: 48%;  
    padding-left: 2%;  
    padding-top: 10px;  
    height: 50px;  
    float: left;  
    background-color: gray;  
    border-radius: 10px;  
    color: white;  
}  
  
#usuario{  
    font-size: 18px;  
    float: left;  
}  
  
#imgusuario{  
    width: 30px;  
    margin-left: 10px;  
    float: left;  
}  
  
#blok1, #blok2{  
    width: 50%;  
    min-height: 20px;  
}
```

```
float: left;
border-radius: 10px;
background-color: gray;
color:white;
text-align: center;
}
section{
width: 100%;
min-height: 300px;
height: auto;
float: left;
margin-top: 5px;
}
.imgastronomia{
width: 90%;
height: 250px;
}
```

### Definiendo la programación JavaScript.

A continuación, definimos la programación en el archivo01.js.

Primero, defina las variables para recuperar la fecha, el día, mes y año, sus valores se visualizarán (innerHTML) en el bloque <div> de id pf, tal como se muestra.

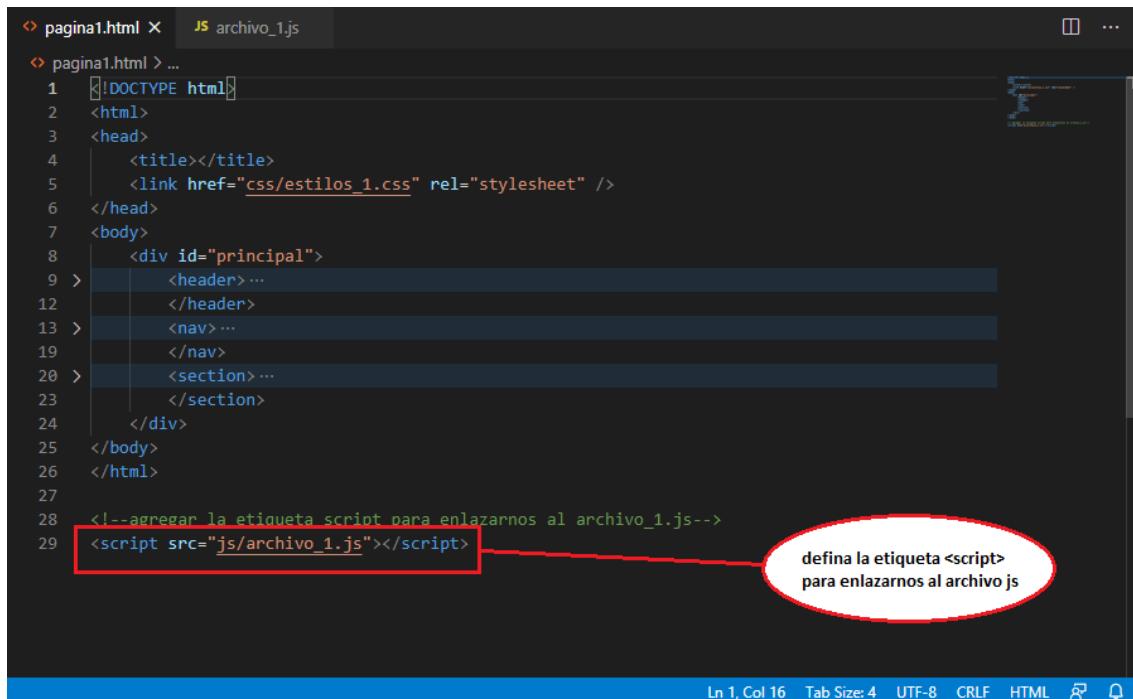


```
pagina1.html JS archivo_1.js ...
js > JS archivo_1.js > ...
1  /*declaramos variables para almacenar la fecha, el
2  dia, mes y año*/
3  let f=new Date();
4  let dia=f.getDate();
5  let mes=f.getMonth()+1;
6  let year=f.getFullYear();
7
8  //en el elemento div cuyo id es pf, imprima el dia, mes y año
9  document.getElementById("pf").innerHTML="Fecha:"+dia+"/"+mes+"/"+year;
10
11 |
```

Ln 11, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ ⚙️

Figura 48 : Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar las operaciones definidas en el archivo \_1.js, defina en la página web la etiqueta <script> para enlazarla al archivo de programación, tal como se muestra.



```
pagina1.html X JS archivo_1.js
pagina1.html > ...
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title></title>
5  |   <link href="css/estilos_1.css" rel="stylesheet" />
6  </head>
7  <body>
8  |   <div id="principal">
9  |       <header>...
12 |       </header>
13 >   <nav>...
19 |   </nav>
20 >   <section>...
23 |       </section>
24   </div>
25 </body>
26 </html>
27
28 <!--agregar la etiqueta script para enlazarnos al archivo_1.js-->
29 <script src="js/archivo_1.js"></script>
```

Figura 49 : Caso práctico

Fuente.- Elaboración Propia

Para ejecutar la página, presiona la tecla F5, donde ejecutamos las sentencias del archivo.js y visualizamos la fecha.

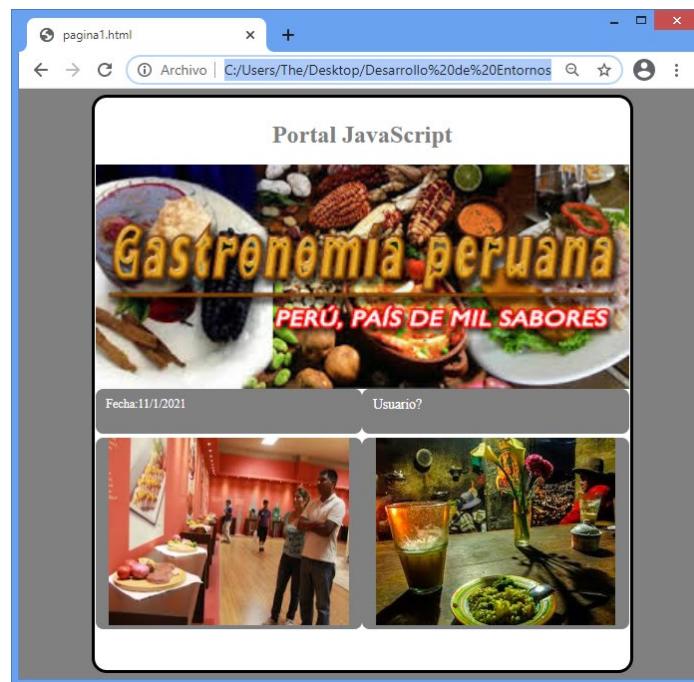
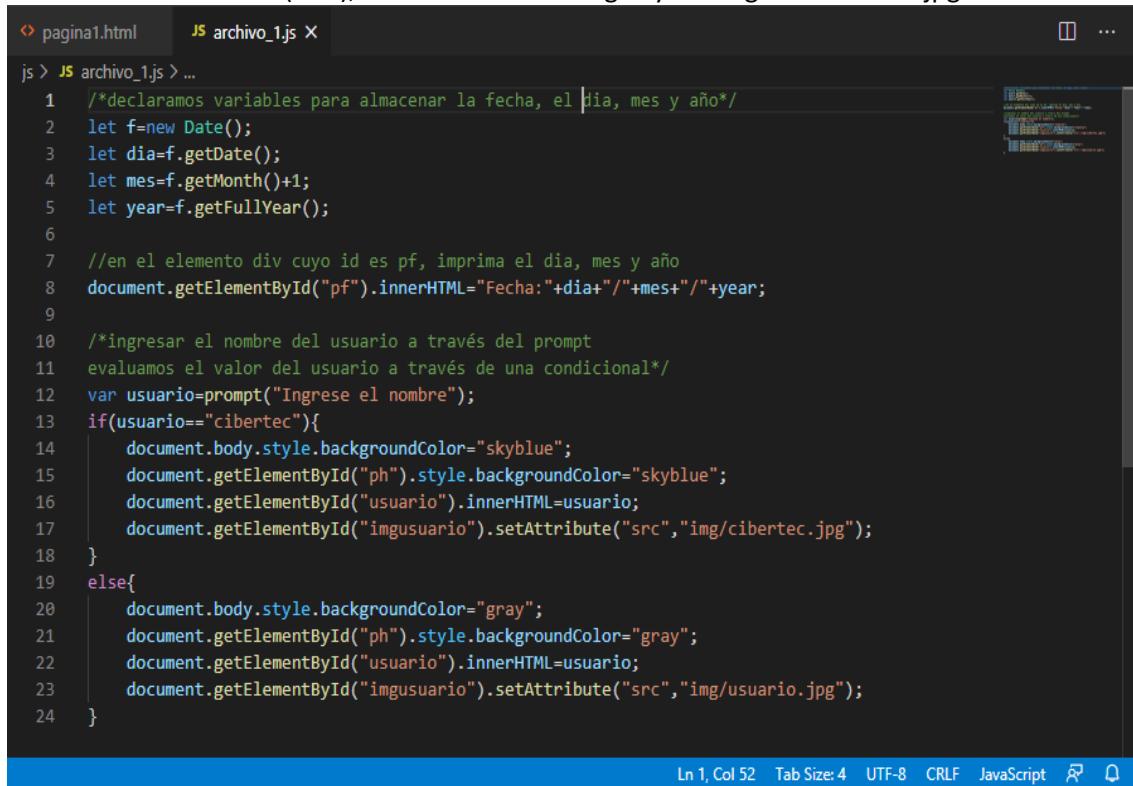


Figura 50: Caso práctico

Fuente.- Elaboración Propia

Definida la primera parte de la programación, vamos a implementar la segunda parte: a través de un prompt ingrese el nombre del usuario.

- Si el valor es “cibertec” el color de fondo es azul y la imagen es cibertec.jpg
- Caso contrario (else), el color de fondo es gris y la imagen es usuario.jpg.



```

pagina1.html      JS archivo_1.js X
js > JS archivo_1.js > ...
1  /*declaramos variables para almacenar la fecha, el dia, mes y año*/
2  let f=new Date();
3  let dia=f.getDate();
4  let mes=f.getMonth()+1;
5  let year=f.getFullYear();
6
7  //en el elemento div cuyo id es pf, imprima el dia, mes y año
8  document.getElementById("pf").innerHTML="Fecha:"+dia+"/"+mes+"/"+year;
9
10 /*ingresar el nombre del usuario a través del prompt
11 evaluamos el valor del usuario a través de una condicional*/
12 var usuario=prompt("Ingrese el nombre");
13 if(usuario=="cibertec"){
14     document.body.style.backgroundColor="skyblue";
15     document.getElementById("ph").style.backgroundColor="skyblue";
16     document.getElementById("usuario").innerHTML=usuario;
17     document.getElementById("imgusuario").setAttribute("src","img/cibertec.jpg");
18 }
19 else{
20     document.body.style.backgroundColor="gray";
21     document.getElementById("ph").style.backgroundColor="gray";
22     document.getElementById("usuario").innerHTML=usuario;
23     document.getElementById("imgusuario").setAttribute("src","img/usuario.jpg");
24 }

```

Ln 1, Col 52 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ 🌐

Figura 51 : Caso práctico  
Fuente.- Elaboración Propia

Cambiar la pestaña a pagina1.html, presiona la tecla F5, donde nos pide ingresar el nombre del usuario; después de ingresar presionar ENTER, donde visualiza: la fecha del sistema y el nombre e imagen del usuario ingresado

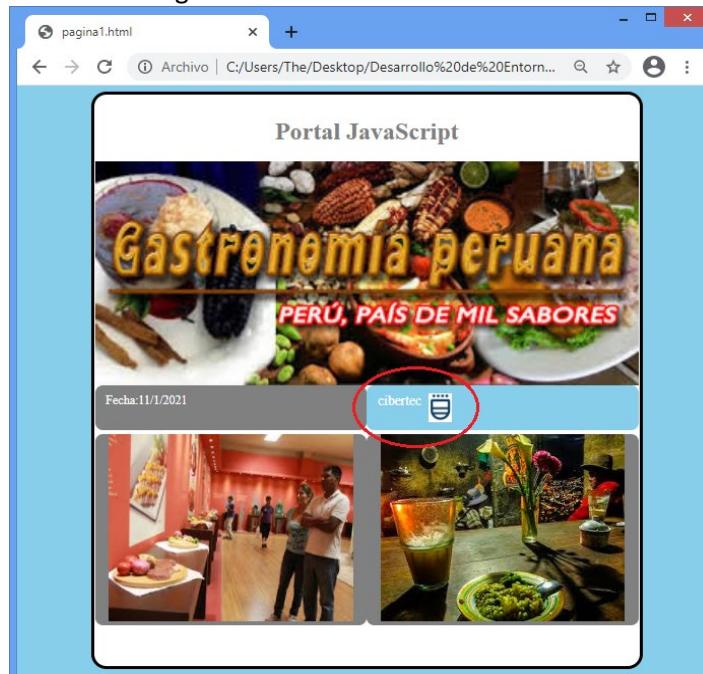


Figura 52 : Caso práctico  
Fuente.- Elaboración Propia

## LABORATORIO 2

### Trabajando con el modelo DOM

Implementa una página web, donde al cargar la página (load), ejecuta las siguientes operaciones:

1. Implementar un carrusel o rotación de imágenes en la imagen de cabecera
2. Imprimir la fecha del sistema
3. Imprimir la hora del sistema y que cambie por intervalos de 1 segundo.

#### 1. Diseñando la página web

En esta etapa vamos a diseñar la página2.html, tal como se muestra



Figura 53 : Caso práctico  
Fuente.- Elaboración Propia

En la página2.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_2.css y al archivo JS: archivo\_2.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

<!-- pagina2.html -->
<!-- pagina2.html > ...
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5      <meta charset="utf-8">
6      <link href="css/estilos_2.css" rel="stylesheet" /> enlazando al archivo css utilizando <link>
7  </head>
8
9  <body>
10 <div id="principal">
11     <header>
12         <h1 id="h1cab">La Gastronomía en el Perú</h1>
13         
14     </header>
15     <nav>
16         <div id="pf">Fecha: ?</div>
17         <div id="ph">Hora: ?</div>
18     </nav>
19     <section>
20         <div class="blok"></div>
21         <div class="blok"></div>
22         <div class="blok"></div>
23         <div class="blok"></div>
24     </section>
25 </div>
26 </body>
27 </html>
28 <script src="js/archivo_3.js"></script> enlazando al archivo js utilizando la etiqueta <script>
29

```

Figura 54 : Caso práctico  
Fuente.- Elaboración Propia

### Definiendo el archivo estilos\_2.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```

body{
    background-color: gray;
}

#principal{
    width: 80%;
    height: auto;
    float: left;
    margin: 0 10% 0 10%;
    border-radius: 20px;
    padding-bottom: 10px;
    background-color: white;
}

header, nav, section, footer{
    width: 100%;
    height: auto;
    float: left;
}

```

```
        margin-bottom: 5px;
    }

#h1cab{
    text-align: center;
    color:gray;
}

#imgcab{
    width: 100%;
    height: 250px;
    float: left;
}

#pf, #ph{
    width: 49%;
    height: 20px;
    float: left;
    padding-left: 1%;
    background-color: gray;
    border-radius: 10px;
    color:white;
}

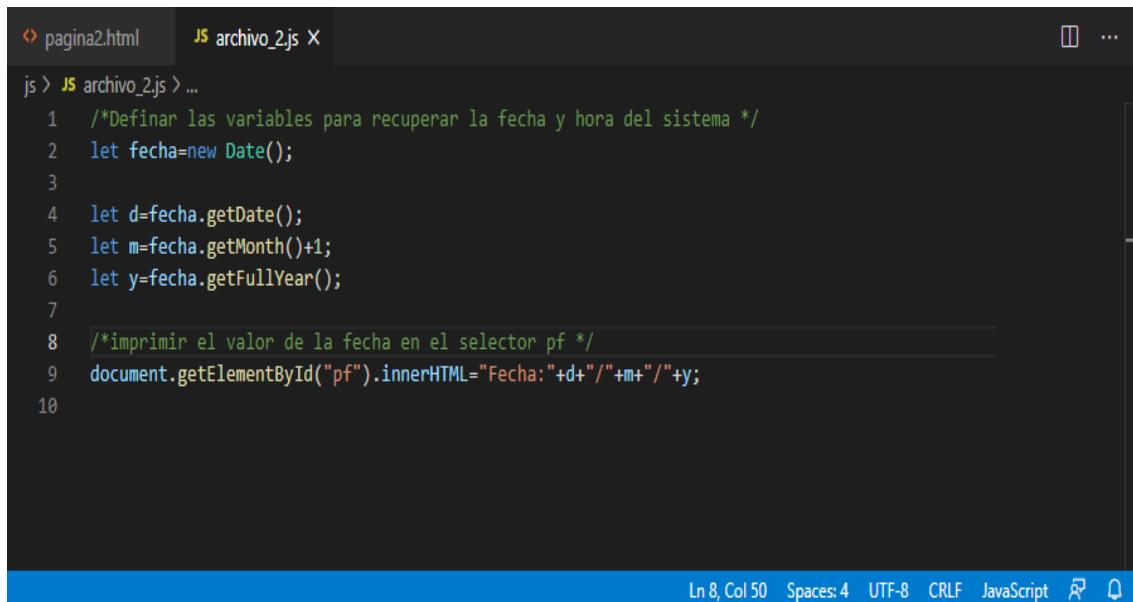
.blok{
    width: 47%;
    height: auto;
    border:1px solid;
    float: left;
    margin:1%;
    background-color: gray;
    color: white;
}

#imgpie{
    width: 100%;
    height: 70px;
    float: left;
    margin-bottom: 20px;
}

.imgblok
{
    width: 100%;
    height: 250px;
    float: left;
}
```

### Definiendo la programación.

1. En el archivo de JavaScript archivo\_2.js, implementamos el proceso para imprimir la fecha, declare variables para almacenar la fecha, el día, mes y año; luego imprima los valores en el selector de id “pf”, tal como se muestra.



```

pagina2.html JS archivo_2.js ...
js > JS archivo_2.js > ...
1  /*Definir las variables para recuperar la fecha y hora del sistema */
2  let fecha=new Date();
3
4  let d=fecha.getDate();
5  let m=fecha.getMonth()+1;
6  let y=fecha.getFullYear();
7
8  /*imprimir el valor de la fecha en el selector pf */
9  document.getElementById("pf").innerHTML="Fecha:"+d+"/"+m+"/"+y;
10

```

Ln 8, Col 50 Spaces: 4 UTF-8 CRLF JavaScript ⚙️

Figura 55 : Caso práctico

Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina2.html, y presiona la tecla F5, donde se visualiza la fecha.

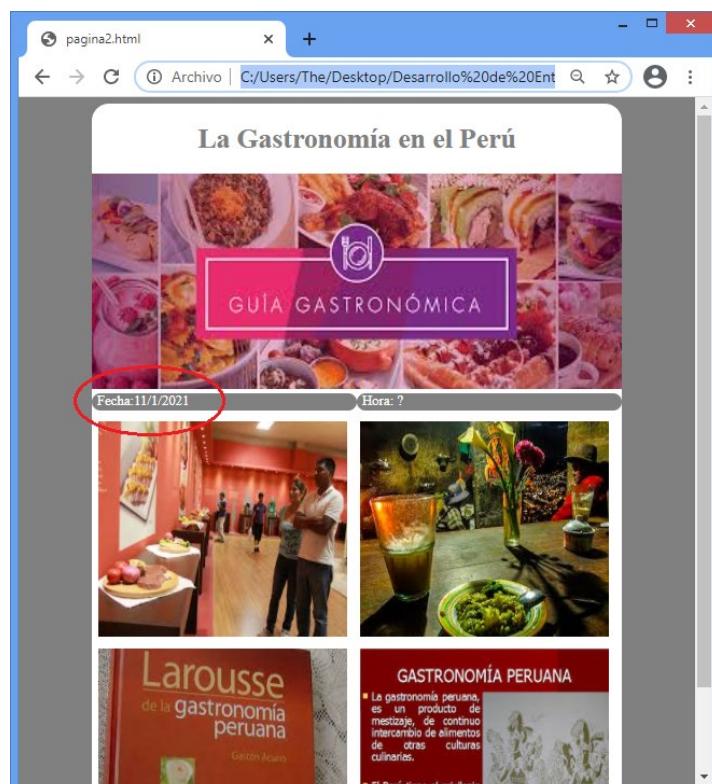


Figura 56: Caso práctico

Fuente.- Elaboración Propia

2. A continuación, implementamos el proceso para imprimir la hora. Defina la función reloj() el cual imprima en el selector de id “ph” la hora por cada segundo, utilizando setTimeout, tal como se muestra.
3. Ejecuta el método asignando al atributo “onload” el método reloj().

```

pagina2.html JS archivo_2.js ...
js > JS archivo_2.js > ...
10
11 /*defina la funcion reloj donde ejecutamos las instrucciones para imprimir la hora
12 por intervalo de 1 segundo, ejecutando setTimeout */
13 function reloj(){
14     let fecha=new Date();
15     let h=fecha.getHours();
16     let m=fecha.getMinutes();
17     let s=fecha.getSeconds();
18
19     //imprimir
20     document.getElementById("ph").innerHTML="Hora:"+h+":"+m+":"+s;
21     setTimeout("reloj()",1000);
22 }
23
24 //al cargar la ventana, ejecuta la funcion
25 document.body.setAttribute("onload","reloj()");

```

Ln 2, Col 22 Spaces: 4 UTF-8 CRLF JavaScript ⚙️ ⚙️

Figura 57 : Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina2.html, y presiona la tecla F5, donde se visualiza la fecha y la hora la cual cambiará por cada segundo.



Figura 58 : Caso práctico  
Fuente.- Elaboración Propia

4. A continuación, implementamos el proceso para cambiar la imagen. Defina la función carrusel() el cual visualiza una nueva imagen en el selector de id "imgcab" por cada segundo, utilizando setTimeout, tal como se muestra.
5. Asignar al atributo “onload” los métodos reloj() y carrusel().

```

pagina2.html JS archivo_2.js ...
js > JS archivo_2.js > ...
23
24 /*defina la funcion carrusel donde ejecutamos las instrucciones para cambiar la imagen
25 de la cabecera (imgcab) intervalo de 1 segundo, ejecutando setTimeout */
26 var c=0;
27 function carrusel(){
28     c++;
29     if(c>5) c=1;
30     document.getElementById("imgcab").setAttribute("src","img/ban"+c+".jpg");
31
32     setTimeout("carrusel()",1000);
33 }
34
35
36 //al cargar la ventana, ejecuta la funcion reloj() y carrusel
37 document.body.setAttribute("onload","reloj(); carrusel()");

```

funcion carrusel()  
donde cambia la  
imagen por cada  
segundo

Asignar el atributo onload, los  
métodos reloj() y carrusel()

Figura 59: Caso práctico

Fuente.- Elaboración Propia

Ejecutar la página2.html, presionando la tecla F5, donde se visualiza la fecha y la hora la cual cambiará por cada segundo y cambiando la imagen de la cabecera.

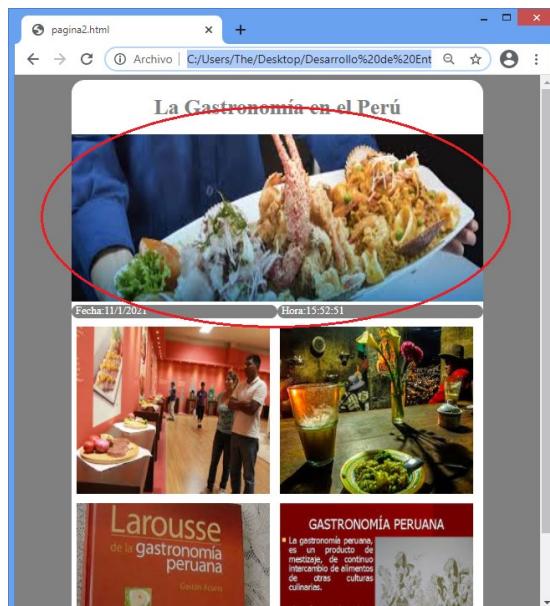


Figura 60 : Caso práctico

Fuente.- Elaboración Propia

## LABORATORIO 3

### Trabajando con el modelo DOM

Implementa una página web, donde al cargar la página (load), visualizamos una ventana de Bienvenida, y al cabo de 5 segundos, la ventana se cierra.

#### 1. Diseñando la página web

En esta etapa vamos a diseñar la página3.html, tal como se muestra. Al cargar la página, se muestra una ventana cuyo título es Bienvenida y al cabo de 5 segundos se cierra.

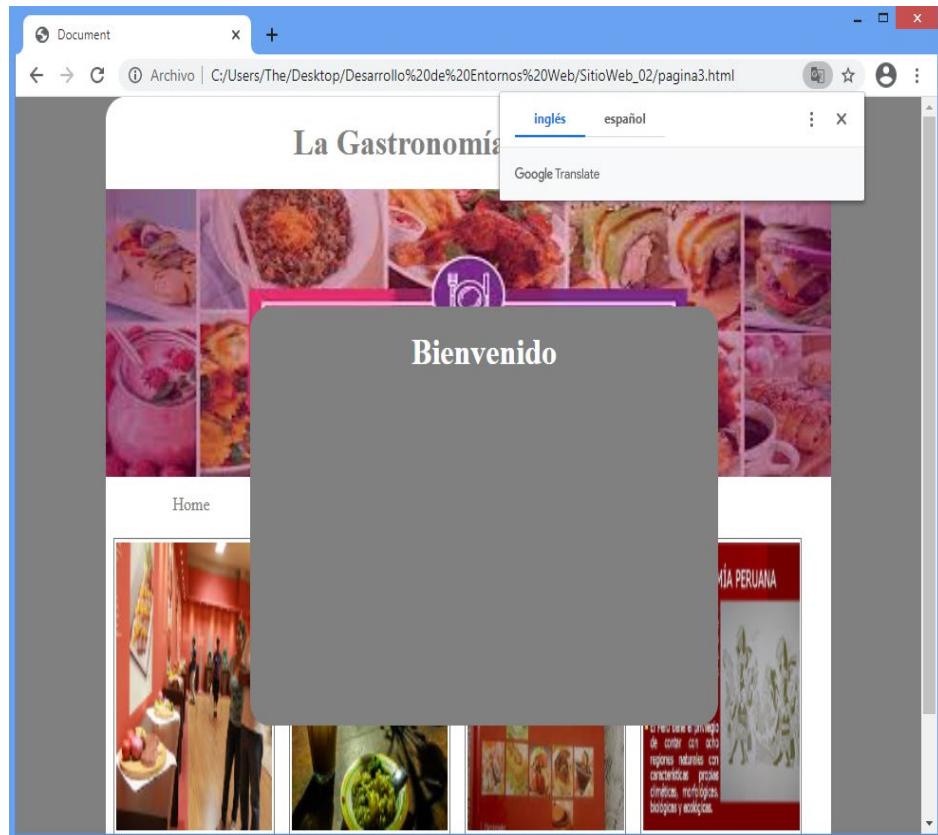


Figura 61: Caso práctico  
Fuente.- Elaboración Propia

En la página3.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_3.css y al archivo JS: archivo\_3.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

![Screenshot of a code editor showing the HTML file 'pagina3.html'. A red box highlights the line <link href=](js/archivo_3.js)

```

1  <meta charset="UTF-8">
2  <meta name="viewport" content="width=device-width, initial-scale=1.0">
3  <link href="css/estilos_3.css" rel="stylesheet">
4
5
6
7
8  <body>
9    <div id="ventana">
10      <h1 id="h1ventana">Bienvenido</h1>
11    </div>
12    <div id="principal">
13      <header>
14        <h1 id="h1cab">La Gastronomía en el Perú</h1>
15        
16      </header>
17      <nav>
18        <a class="item" href="#">Home</a>
19        <a class="item" href="#">Historia</a>
20        <a class="item" href="#">Gastronomía</a>
21        <a class="item" href="#">Platos Bandera</a>
22        <a class="item" href="#">Restaurantes</a>
23        <a class="item" href="#">Contactenos</a>
24      </nav>
25      <section>
26        <div class="blok"></div>
27        <div class="blok"></div>
28        <div class="blok"></div>
29        <div class="blok"></div>
30      </section>
31      <footer>
32        
33      </footer>
34    </div>
35  </body>
36 </html>
37 <script src="js/archivo_3.js"></script>

```

Figura 62 : Caso práctico

Fuente.- Elaboración Propia

### Definiendo el archivo estilos\_3.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```

body{
  background-color: gray;
  margin: 0px auto;
}

#principal{
  width: 80%;
  height: auto;
  float: left;
  margin: 0 10% 0 10%;
  border-radius: 20px 20px 0 0;
  background-color: white;
}

header, section, footer{
  width: 100%;
  height: auto;
  float: left;
  margin-bottom: 5px;
}

nav{

```

```
width: 90%;  
margin:0 5% 0 5%;  
height: auto;  
float: left;  
text-align:right;  
}  
.item{  
width:15%;  
margin:10px 0 10px 0;  
height:20px;  
float:left;  
color:gray;  
text-align:center;  
text-decoration:none;  
}  
.item:hover{  
color:blue;  
text-decoration:underline;  
}  
#h1cab{  
text-align: center;  
color:gray;  
}  
#imgcab{  
width: 100%;  
height: 250px;  
float: left;  
margin:0;  
opacity: 1;  
}  
.h2cab {  
text-align: center;  
color: gray;  
}  
.blok{  
width: 22%;  
min-height: 200px;  
height: auto;  
border:1px solid;  
float: left;  
margin:1%;  
color: gray;  
}  
.imgblok{  
width:98%;  
height:250px;  
margin:3px 1% 3px 1%;  
float:left;  
}  
#imgpie{  
width: 100%;  
height: 70px;
```

```

        float: left;
        margin-top: 20px;
    }
    #ventana{
        position: absolute;
        border-radius: 20px;
        background-color: gray;
        color: white;
        top: 0px;
        left: 0px;
        width: 0px;
        height: 0px;
    }
    #h1ventana{
        text-align: center;
    }
}

```

### Definiendo la programación.

1. En el archivo de JavaScript archivo\_3.js, implementamos una función “onload” de **Window** para visualizar el elemento <div> de id “ventana”.



```

pagina3.html JS archivo_3.js ...
js > JS archivo_3.js ...
1 window.onload = function() {
2
3
4
5
6
7
8

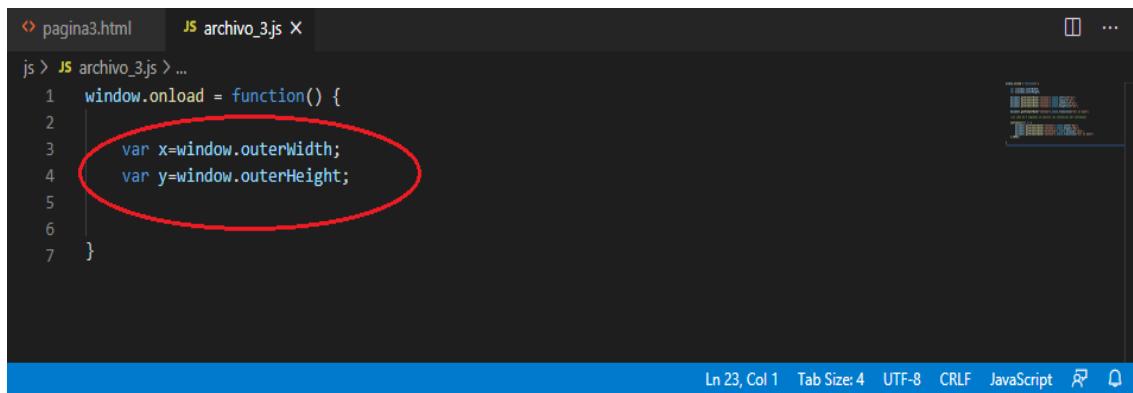
```

Ln 23, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ ⌂

Figura 63: Caso práctico

Fuente.- Elaboración Propia

2. Declare dos variables las cuales almacenan el espacio, en píxeles, que está ocupando la ventana del navegador: outerWidth y outerHeight (Window)



```

pagina3.html JS archivo_3.js ...
js > JS archivo_3.js ...
1 window.onload = function() {
2
3     var x=window.outerWidth;
4     var y=window.outerHeight;
5
6
7

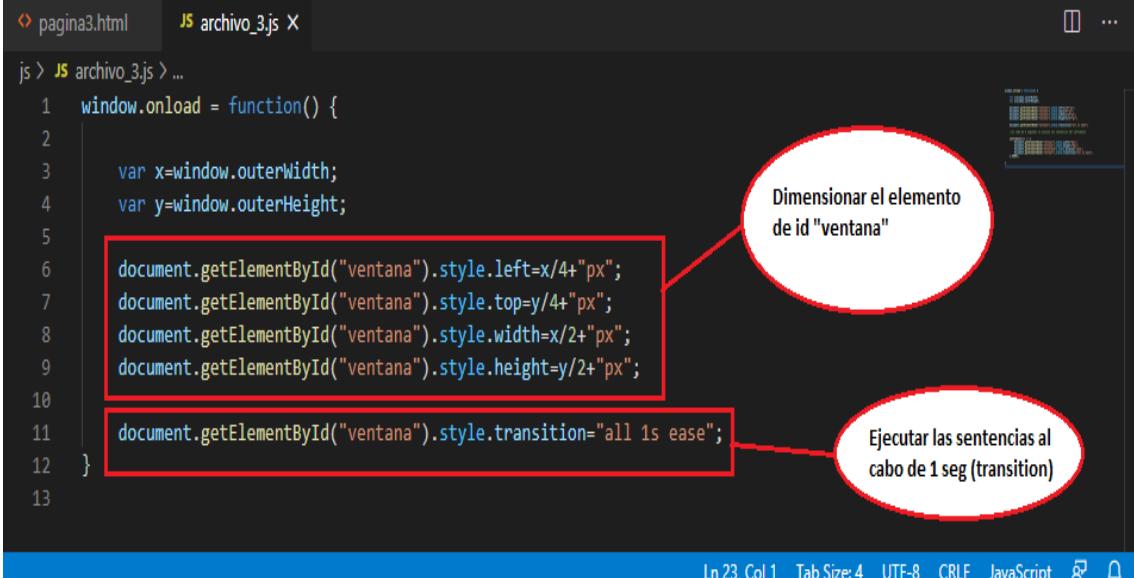
```

Ln 23, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ ⌂

Figura 64 : Caso práctico

Fuente.- Elaboración Propia

3. Con las variables definidas, dimensionar el elemento “ventana” utilizando las propiedades: left, top, width y height (aplicamos por ser un elemento de position absolute)
4. Para que las sentencias denoten un efecto, ejecutamos un transition de todas las sentencias en 1 segundo.



The screenshot shows a code editor with two tabs: 'pagina3.html' and 'JS archivo\_3.js'. The code in 'archivo\_3.js' is:

```

1 window.onload = function() {
2
3     var x=window.outerWidth;
4     var y>window.outerHeight;
5
6     document.getElementById("ventana").style.left=x/4+"px";
7     document.getElementById("ventana").style.top=y/4+"px";
8     document.getElementById("ventana").style.width=x/2+"px";
9     document.getElementById("ventana").style.height=y/2+"px";
10
11    document.getElementById("ventana").style.transition="all 1s ease";
12 }

```

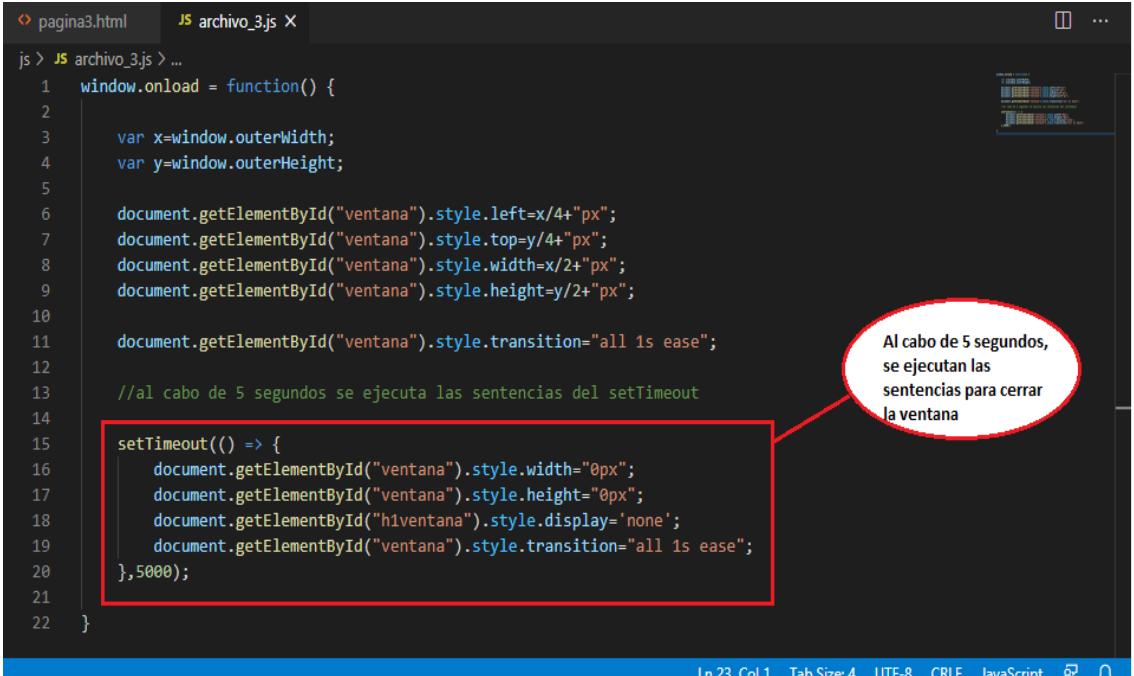
Annotations with red boxes highlight the following sections:

- A box around lines 6-9 is labeled "Dimensionar el elemento de id 'ventana'" (Resize the element with id "ventana").
- A box around line 11 is labeled "Ejecutar las sentencias al cabo de 1 seg (transition)" (Execute the statements after 1 second (transition)).

At the bottom of the editor, status bars show: Ln 23, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚡ 🌐

Figura 65 : Caso práctico  
Fuente.- Elaboración Propia

- Para finalizar el proceso, definimos un setTimeout(), donde al cabo de 5 segundos, deberá cerrar la ventana.



The screenshot shows a code editor with two tabs: 'pagina3.html' and 'JS archivo\_3.js'. The code in 'archivo\_3.js' is:

```

1 window.onload = function() {
2
3     var x=window.outerWidth;
4     var y>window.outerHeight;
5
6     document.getElementById("ventana").style.left=x/4+"px";
7     document.getElementById("ventana").style.top=y/4+"px";
8     document.getElementById("ventana").style.width=x/2+"px";
9     document.getElementById("ventana").style.height=y/2+"px";
10
11    document.getElementById("ventana").style.transition="all 1s ease";
12
13    //al cabo de 5 segundos se ejecutan las sentencias del setTimeout
14
15    setTimeout(() => {
16        document.getElementById("ventana").style.width="0px";
17        document.getElementById("ventana").style.height="0px";
18        document.getElementById("h1ventana").style.display='none';
19        document.getElementById("ventana").style.transition="all 1s ease";
20    },5000);
21
22 }

```

Annotations with red boxes highlight the following sections:

- A box around the setTimeout block (lines 15-20) is labeled "Al cabo de 5 segundos, se ejecutan las sentencias para cerrar la ventana" (After 5 seconds, the statements to close the window are executed).

At the bottom of the editor, status bars show: Ln 23, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚡ 🌐

Figura 66 : Caso práctico  
Fuente.- Elaboración Propia

Ejecutar la página3.html, presionando la tecla F5, donde se visualiza la ventana de bienvenida y al cabo de 5 segundos se cerrará.

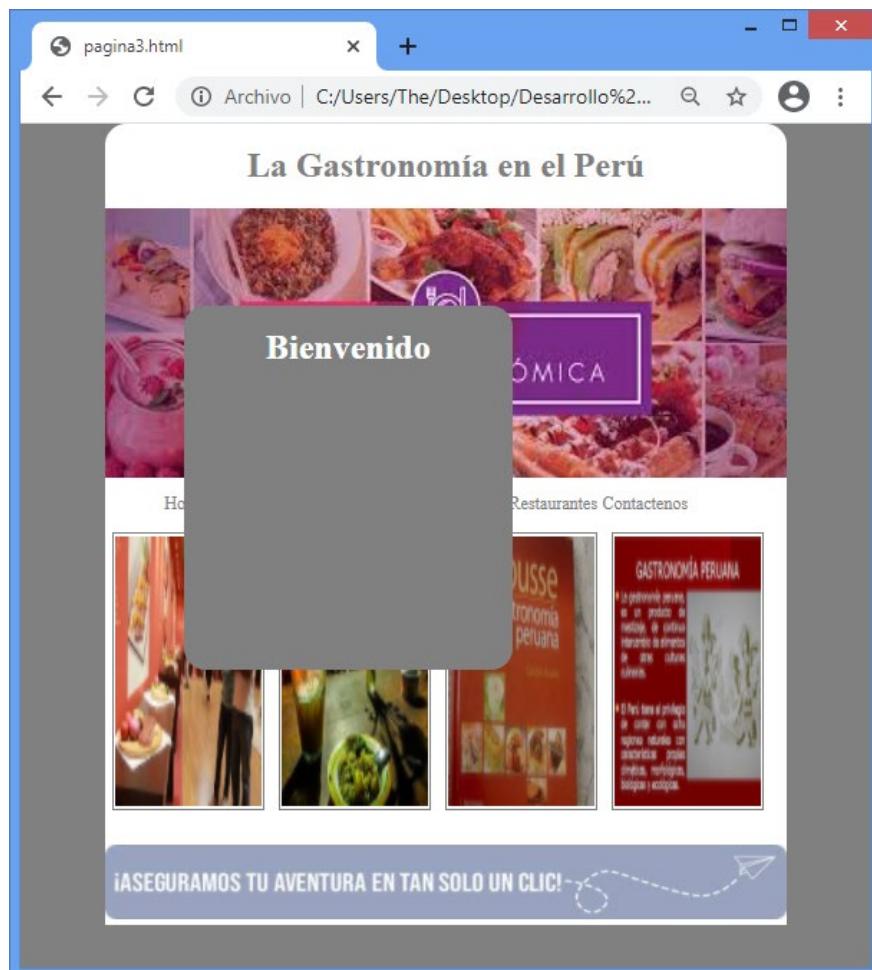


Figura 67: Caso práctico  
Fuente.- Elaboración Propia

# Resumen

1. El Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API) para documentos válidos HTML y bien construidos XML que define la estructura lógica de los documentos y el modo en que se accede y manipula.
2. El DOM permite un acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto.
3. Los métodos que permiten acceder a los elementos de una página HTML son: getElementById(), getElementsByTagName(), getElementsByClassName(), querySelector(), querySelectorAll().
4. En HTML todas las etiquetas son elementos, tales como <p>, <img> y <div> por lo que tienen atributos y contienen nodos hijos. Sin embargo, los nodos de textos no poseen atributos e hijos.
5. Los atributos que le son asignados a las etiquetas HTML están disponibles como propiedades de sus correspondientes nodos en el DOM. La función getAttribute() retorna el valor de atributo, la función setAttribute() permite asignar un valor a un atributo de un elemento. Las propiedades de estilo pueden ser aplicadas a través del DOM.
6. Cada atributo CSS posee una propiedad del DOM equivalente, formándose con el mismo nombre del atributo CSS, pero sin los guiones y llevando la primera letra de las palabras a mayúsculas.

# Recursos

Pueden revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [http://librosweb.es/libro/JavaScript/capitulo\\_5/ejercicios\\_sobre\\_dom.html](http://librosweb.es/libro/JavaScript/capitulo_5/ejercicios_sobre_dom.html)
- [http://librosweb.es/libro/ajax/capitulo\\_4.html](http://librosweb.es/libro/ajax/capitulo_4.html)
- <http://www.laweberta.es/como-hacer/ejemplos-css/cambiar-estilo-css-web-dinamicamente-i.php>
- <http://lineadecodigo.com/JavaScript/crear-elementos-html-con-JavaScript/>
- <https://uniwebsidad.com/libros/JavaScript/capitulo-5/acceso-directo-a-los-nodos>
- <https://developer.mozilla.org/es/docs/Web/API/Document>



# FUNCIONES Y EVENTOS

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno diseña y construye páginas para un sitio web aplicando el modelo de objetos de documento DOM, definiendo e implementando funciones y eventos.

## TEMARIO

### 3.1 Tema 3 : Manejo de funciones

- 3.1.1 : Introducción
- 3.1.2 : Sintaxis general
- 3.1.3 : Funciones y argumentos
- 3.1.4 : Funciones con return
- 3.1.5 : Manejo del setTimeout() y setInterval()

### 3.2 Tema 4 : Eventos

- 3.2.1 : Introducción
- 3.2.2 : Tipos de eventos
- 3.2.3 : Manejadores de eventos
- 3.2.4 : Manejo de listener

## ACTIVIDADES PROPUESTAS

- Mostrar la fecha del sistema.
- Mostrar una ventana de diálogo emergente.
- Mostrar un reloj.

### 3.1. MANEJO DE FUNCIONES

#### 3.1.1 Introducción

Una función no es más que un bloque de enunciados que componen un comportamiento que puede ser invocado las veces que sea necesario.

#### 3.1.2 Sintaxis general

Una función de JavaScript presenta este aspecto:

```
function nombre_de_la_función(){
    ...enunciados a ejecutar...
}
```

Para ejecutar la función posteriormente no hay más que invocar su nombre en cualquier momento y desde cualquier parte de un código, con una excepción: la función debe haber sido definida anteriormente. Así, por ejemplo, este código:

```
function dame_una_a(){
    alert("¡AAAAAAAAAAAAAAA!");
}

dame_una_a();
```

Ejecutaría la alerta, pero éste:

```
dame_una_a();

function dame_una_a(){
    alert("¡AAAAAAAAAAAAAAA!");
}
```

Generaría un error, porque en el momento en que se invoca la función ésta aún no ha sido registrada.

Hay que poner especial atención a la hora de crear las funciones, para no repetir los nombres, principalmente porque esto no genera errores en JavaScript, y puede suponer dolores de cabeza cuando un script no funciona, pero la consola de errores no muestra mensaje alguno.

Si definimos dos funciones con el mismo nombre, como en este ejemplo:

```
function mensaje(){
    alert("hola que tal");
}
function mensaje(){
    alert("hola que tales saludos a todos");
}
```

Sólo funciona la segunda, que ha sido la última definida.

En el siguiente ejemplo se muestra el uso de funciones para cambiar una imagen por otra al pasar el mouse encima.

Cuando ponemos el puntero del mouse en el enlace se cargará la imagen2 y cuando el puntero sale del enlace, regresará la imagen original imagen1.

```
<HTML>
<HEAD>
<TITLE>cambia imagen</TITLE>
<META charset="utf-8">
<SCRIPT>
function cambiar () {
    document.images["modelo"].src = " imagen2.jpg";
}
function volver () {
    modelo.src = " imagen1.jpg";
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="#" onMouseOver="cambiar();" onMouseOut="volver();">Pasa el mouse
sobre mí y cambiará la imagen</A><BR><BR><BR><IMG src="imagen1.jpg"
NAME="modelo">
</BODY>
</HTML>
```

### 3.1.3 Funciones y argumentos

Podemos desear que una función ejecute unos enunciados en los que opere con una serie de valores que no hayamos definido dentro de la misma, sino que los reciba de otra función o enunciado. Esos valores son los argumentos o parámetros, que se especifican entre los paréntesis que van tras el nombre de la función, y se separan por comas:

```
function nombre_de_la_función(argumento1,argumento1){
    ...enunciados a ejecutar...
}
```

Los argumentos se nombran las variables:

```
function sumar(x,y){
    var total = x + y;
    alert(total);
}
```

Si después se ejecuta unas líneas como las siguientes:

```
sumar(1,2);
sumar(3,5);
sumar(8,13);
```

En la función sumar total adquiere sucesivamente los valores de 3, 8 y 21, que es lo que mostrarían tres alertas.

Aunque en los ejemplos emplee numerales, como argumentos se puede enviar cualquier variable. Sólo hay que recordar que, si se trata de una cadena literal, debe ir entrecomillada:

```
la_función('cadena','otra_cadena');
```

Por último, sobre los argumentos hay que recordar las respuestas a tres preguntas:

- ¿Qué ocurre si se envía a una función menos argumentos que los que se han especificado?: Los argumentos que no han recibido un valor adoptan el de undefined.
- ¿Qué ocurre si se envía a una función más argumentos que los que se han especificado?: Los argumentos que sobran son ignorados.
- ¿Cuántos argumentos se pueden especificar como máximo?: 255.

Vamos a crear una función que realice cuatro operaciones aritméticas con los valores que introduzca un usuario. Se podría crear una función para cada operación, pero para simplificar el código es mejor crear una sola que obtenga los valores de los campos de formulario con los valores con los que operar, y que luego realice la operación elegida. ¿Y cómo sabe la función qué operación realizar? Eso es justo lo que pasamos como argumento.

La función sería ésta:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>funciones con argumentos</title>

<script>
    function operar(x){
        var valor_01 = eval(document.getElementById('campo_01').value);
        var valor_02 = eval(document.getElementById('campo_02').value);
        switch(x){
            case('sumar'):
                var resultado = valor_01 + valor_02;
                break;
            case('restar'):
                var resultado = valor_01 - valor_02;
                break;
            case('multiplicar'):
                var resultado = valor_01 * valor_02;
                break;
            case('dividir'):
                var resultado = valor_01 / valor_02;
                break;
        }
        document.getElementById('total').value = "Valor de "+x+" "+valor_01+" y "+valor_02+" es "+resultado+"!";
    }
</script>
```

```

</head>
<body>
<textarea id="total"></textarea>
Ingrese un numero <input id="campo_01" />
Ingrese otro numero <input id="campo_02" />

<button onclick="operar('sumar');">+</button>
<button onclick="operar('restar');">-</button>
<button onclick="operar('multiplicar');">x</button>
<button onclick="operar('dividir');">/</button>

</body>
</html>

```

Hacer una redirección con JavaScript es muy sencillo. Podemos hacer que la redirección actúe justo al cargar la página o que actúe tras un cierto tiempo.

Redirección al cargar la página:

```
<body onLoad="document.location.href='http://www.cibertec.edu.pe'">
```

Si ponemos esto en nuestra etiqueta "body", cuando la página cargue, redireccionará a la página web de Cibertec. Si nos interesa más una redirección que sea efectiva transcurridos unos segundos, podemos hacerlo así:

Entre <head> y </head> pondremos:

```

<script type="text/JavaScript">
var pagina = 'http://www.cibertec.edu.pe';
var segundos = 5;
function redireccion() {
    document.location.href=pagina;
}
setTimeout("redireccion()",segundos);
</script>

```

Tan solo tenemos que cambiar las variables página y segundos, marcadas en negrita, para que redirija a la página que queramos y en el tiempo que queramos.

Lo que hacemos es crear un "timeout" que llame a la función "redireccion()" transcurridos tantos segundos como marque la variable "segundos". La función redirección lo único que hace es redirigir a la página que se indica en la variable "pagina".

### 3.1.4 Funciones con return

Una función con return es un módulo de programa que puede recibir datos de entrada a través de variables locales denominadas argumentos y que retorna un resultado al punto donde es invocado. Este tipo de función se utiliza para efectuar cualquier tipo de proceso que produzca un resultado

La función anterior llamada concatenar\_cadenas, se puede modificar para que retorne un valor, de la siguiente manera:

```
function concatenar_cadenas(){
    var a = "orda";
    var b = "lía";
    var c = a + b;
    return c;
}
```

```
function mostrar_resultado(){
    alert(concatenar_cadenas());
}
```

```
mostrar_resultado();
```

### Ejemplo 1: Deshabilita la selección de textos con el mouse

En el siguiente ejemplo, se muestra el uso de las funciones con return para deshabilitar la selección de textos con el mouse:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
</head>
<body>
Hola que tal, intenta seleccionar este texto
<script language="JavaScript">
<!-- Begin
function disableselect(e){
return false
}
function reEnable(){
return true
}
document.onselectstart=new Function ("return false" )
if (window.sidebar){
document.onmousedown=disableselect
document.onclick=reEnable
}
// End --&gt;
&lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

### Ejemplo 2: Mostrar fecha

Confeccionar una función que reciba una fecha con el formato de día, mes y año y retorne un string con un formato similar a: "Hoy es 20 de enero de 2021".

```

<html>
<head>
</head>
<body>
<script type="text/JavaScript">

function formatearFecha(dia,mes,año)
{
    var s='Hoy es '+dia+' de ';
    switch (mes) {
        case 1:s=s+'enero '; break;
        case 2:s=s+'febrero ';break;
        case 3:s=s+'marzo '; break;
        case 4:s=s+'abril '; break;
        case 5:s=s+'mayo '; break;
        case 6:s=s+'junio '; break;
        case 7:s=s+'julio '; break;
        case 8:s=s+'agosto '; break;
        case 9:s=s+'septiembre '; break;
        case 10:s=s+'octubre '; break;
        case 11:s=s+'noviembre '; break;
        case 12:s=s+'diciembre '; break;
    } //fin del switch
    s=s+'de '+año;
    return s;
}

document.write(formatearFecha(20,1,2021));

</script>
</body>
</html>

```

Analicemos un poco la función formatearFecha. Llegan tres parámetros con el día, mes y año. Definimos e inicializamos una variable con:

```
var s='Hoy es '+dia+' de ';
```

Luego le concatenamos o sumamos el mes:

```
s=s+'enero ';
```

Esto, si el parámetro mes tiene un uno. Observemos como acumulamos lo que tiene 's' más el string 'enero'. En caso de hacer s='enero ' perderíamos el valor previo que tenía la variable s.

Por último, concatenamos el año:

```
s=s+'de '+año;
```

Cuando se llama a la función directamente, al valor devuelto se lo enviamos a la función write del objeto document. Esto último lo podemos hacer en dos pasos:

```
var fec= formatearFecha(11,6,2013);
document.write(fec);
```

Guardamos en la variable 'fec' el string devuelto por la función.

### Ejemplo 3: Cajas de dialogo personalizadas

En lugar de usar los cuadros de alerta con el método alert, se pueden crear cajas de dialogo personalizadas:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<style>
#dialogoverlay{
    display: none;
    opacity: .8;
    position: fixed;
    top: 0px;
    left: 0px;
    background: #FFF;
    width: 100%;
    z-index: 10;
}
#dialogbox{
    display: none;
    position: fixed;
    background: #000;
    border-radius:7px;
    width:550px;
    z-index: 10;
}
#dialogbox > div{ background:#FFF; margin:8px; }
#dialogbox > div > #dialogboxhead{ background: #666; font-size:19px; padding:10px;
color:#CCC; }
#dialogbox > div > #dialogboxbody{ background:#333; padding:20px; color:#FFF; }
#dialogbox > div > #dialogboxfoot{ background: #666; padding:10px; text-align:right; }
</style>

<script>
function CustomAlert(){
    this.render = function(dialog){
        var winW = window.innerWidth;
        var winH = window.innerHeight;
        var dialogoverlay = document.getElementById('dialogoverlay');
        var dialogbox = document.getElementById('dialogbox');
        dialogoverlay.style.display = "block";
        dialogoverlay.style.height = winH+"px";
        dialogbox.style.left = (winW/2) - (550 * .5)+"px";
        dialogbox.style.top = "100px";
    }
}
```

```
dialogbox.style.display = "block";
document.getElementById('dialogboxhead').innerHTML = "Acknowledge This
Message";
document.getElementById('dialogboxbody').innerHTML = dialog;
document.getElementById('dialogboxfoot').innerHTML = '<button
onclick="Alert.ok()">OK</button>';
}
this.ok = function(){
    document.getElementById('dialogbox').style.display = "none";
    document.getElementById('dialogoverlay').style.display = "none";
}
}
var Alert = new CustomAlert();
</script>

</head>
<body>
<div id="dialogoverlay"></div>
<div id="dialogbox">
<div>
<div id="dialogboxhead"></div>
<div id="dialogboxbody"></div>
<div id="dialogboxfoot"></div>
</div>
</div>
<h1>My web document content ...</h1>
<h2>My web document content ...</h2>
<button onclick="alert('You look very pretty today.')">Default Alert</button>
<button onclick="Alert.render('You look very pretty today.')">Custom Alert</button>
<button onclick="Alert.render('And you also smell very nice.')">Custom Alert 2</button>
<h3>My web document content ...</h3>
</body>
</html>
```

### 3.1.5. Manejo del setTimeout() y setInterval()

En sus inicios, JavaScript era un lenguaje que servía principalmente para validar campos en formularios. Incluso muchos programadores de la década de los 90' no terminaban de considerarlo un lenguaje serio.

La caída de Netscape, que lo inventó, frente al dominio del Explorer de Microsoft hizo que durante unos años su desarrollo quedara en la sombra. Sin embargo, con el tiempo fue ganando en riqueza y complejidad. Y de aquí su resurgir gracias a empresas como Google y Yahoo en la década de los 2000.

Lo que hoy nos trae por aquí es el uso de una de estas funcionalidades que permite hacer maravillas. Se trata de las funciones setInterval() y setTimeOut() que usaremos para eventos programados y animaciones.

## Función setInterval( )

```
setInterval(nombreFuncion, intervaloMilisegundos)
```

Este es uno de los eventos cronometrados. El objeto de Window permite que el código se ejecute en cada intervalo de tiempo determinado. Este objeto ha proporcionado setInterval () para repetir una función cada cierto tiempo.

Toma dos parámetros como argumentos. Uno es la función y el otro es el tiempo que especifica el intervalo después del cual la función debe repetirse.

### Forma 1 de escribirlo

```
setInterval(saludar, 5000);  
  
function saludar()  
{  
    alert("Hola");  
}
```

Y cada 5s ejecutaríamos la función saludar que ha quedado declarada en la segunda línea de código y que hemos asignado y escrito sin el paréntesis ( ).

En algunas bibliografías muy conocidas de internet también lo encontraréis todo en una línea como:

### Forma 2 de escribirlo

```
setInterval(function(){alert("Hola");},5000);
```

Que directamente no usa el nombre la función, pero asigna una funcionalidad. A esto se llama **función anónima**.

El método setInterval () continuará llamando a la función hasta que se llame a clearInterval () o se cierre la ventana.

El valor de ID devuelto por setInterval () se usa como parámetro para el método clearInterval ().

Para ejecutar una función solo una vez, después de un número específico de milisegundos, use el método **setTimeout ()**.

## Función setTimeout()

```
setTimeout(nombreFuncion, intervaloMilisegundos)
```

Este es uno de los eventos cronometrados. El objeto de **Window** permite la ejecución de código en intervalos de tiempo específicos. Este objeto ha proporcionado setTimeout () para ejecutar una función después de una cierta cantidad de tiempo.

Toma dos parámetros como argumentos. Uno es la función y el otro es el tiempo que especifica el intervalo después del cual se debe ejecutar la función.

En el siguiente ejemplo, el tiempo transcurrido hasta la función setTimeout () es de 3 segundos. Por lo tanto, la función se ejecutará después de tres segundos y mostrará la salida como se muestra:

```
<html>
<body>
<p id = "time"></p>
<script>
setTimeout(function(){
    document.getElementById("time").innerHTML = "Cibertec es el Instituto Tecnológico
número 1 del país"; }, 3000);
</script>
</body>
</html>
```

## LABORATORIO 1

### Trabajando con funciones

Implementa funciones para dar animaciones a los elementos de la cabecera de una página web:

- En el título <h1> de la cabecera de la página debe cambiar de color por cada segundo.
- En la imagen <img> de la cabecera de la página, debe “desvanecer” y “aparecer” por cada segundo.

### 1. Creando el Sitio Web

Defina una carpeta llamado SitioWeb\_03, en ella crea las siguientes subcarpetas, tal como se muestra.

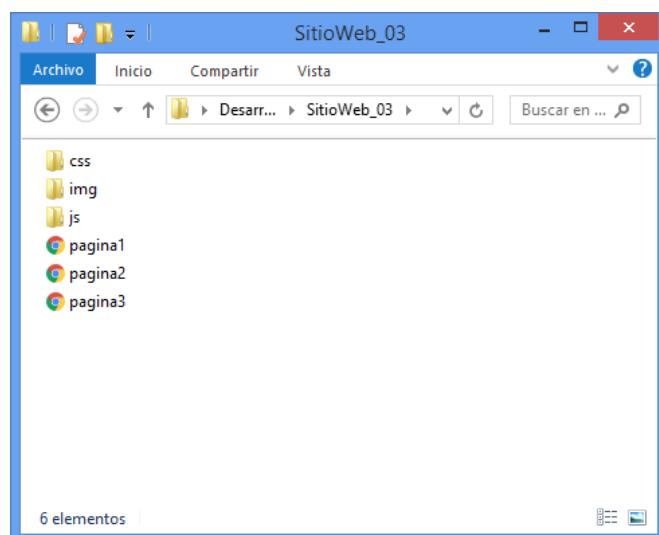


Figura 68 : Caso práctico  
Fuente.- Elaboración Propia

### 2. Trabajando con Visual Studio Code

Para crear una página web, abrir el aplicativo Visual Studio Code.

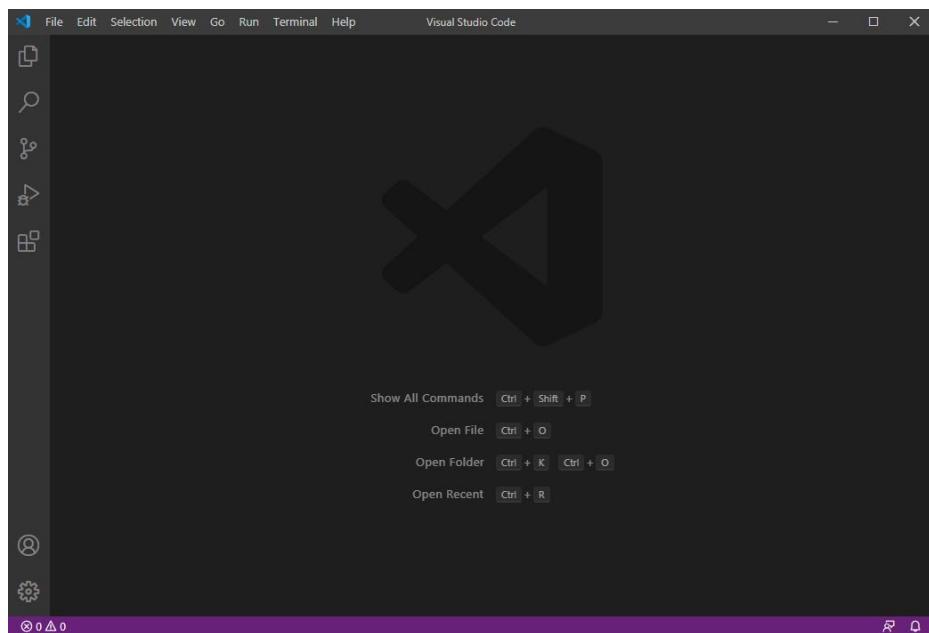


Figura 69 : Caso práctico  
Fuente.- Elaboración Propia

Para trabajar con la carpeta llamada SitioWeb\_03 en Visual Studio Code, arrastre la carpeta hacia la ventana del Visual, tal como se muestra; y luego soltarla.

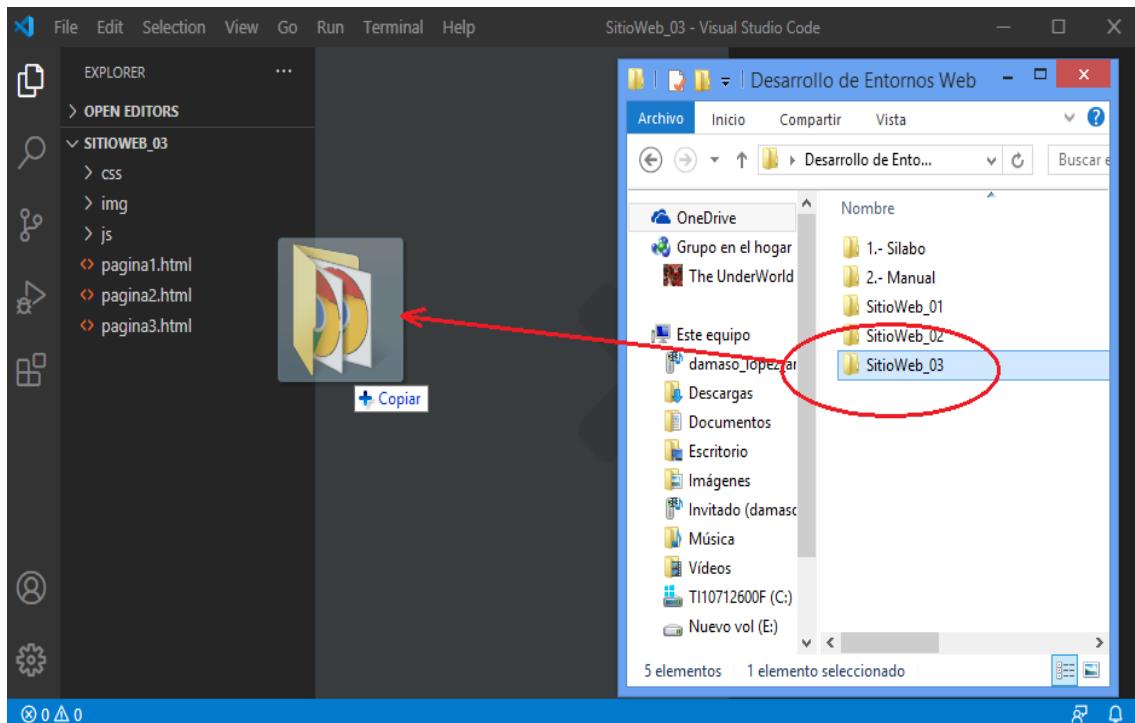


Figura 70 : Caso práctico  
Fuente.- Elaboración Propia

Donde se visualiza la carpeta en la parte izquierda de la ventana del aplicativo

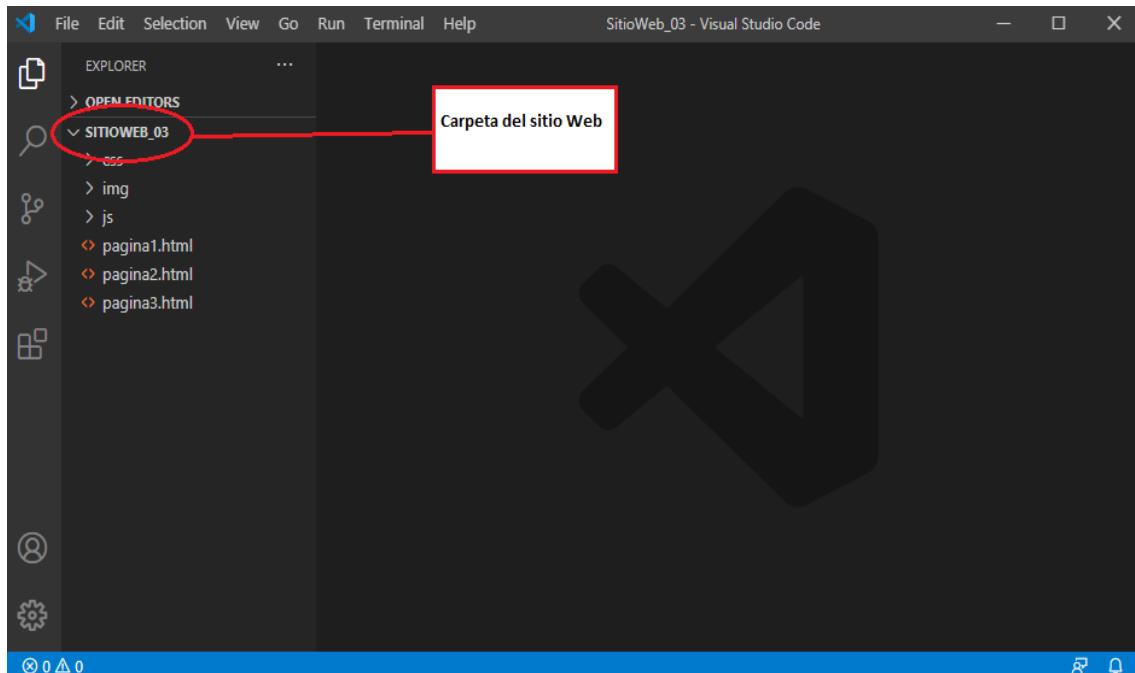


Figura 71: Caso práctico  
Fuente.- Elaboración Propia

## 1. Diseñando la página web

En esta etapa vamos a diseñar la página1.html, tal como se muestra

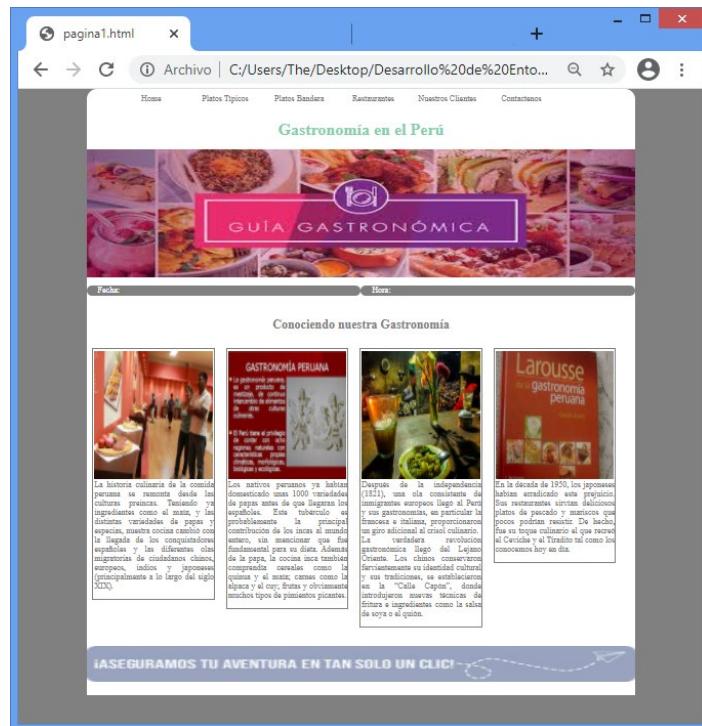


Figura 72: Caso práctico  
Fuente.- Elaboración Propia

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_1.css y al archivo JS: archivo\_1.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

▷ pagina1.html JS archivo_1.js
▷ pagina1.html > ...
1  !DOCTYPE html
2  <html>
3  <head>
4      <title></title>
5      <meta charset="utf-8">
6      <link href="css/estilos_1.css" rel="stylesheet" />
7  </head>
8  <body>
9      <div id="principal">
10     <nav>...
11     </nav>
12     <header>...
13     </header>
14     <section>...
15     </section>
16     <footer>
17         
18     </footer>
19     </div>
20 </body>
21 </html>
22 <script src="js/archivo_1.js"></script>
23
24
25
26
27

```

Ln 1, Col 16 Tab Size: 4 UTF-8 with BOM CRLF HTML ⚙ ⚙

Enlazando <link> al archivo css.

Enlazando <script> al archivo js

Figura 73: Caso práctico  
Fuente.- Elaboración Propia

## Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```

<body>
    <div id="principal">
        <nav>
            <a class="item" href="#">Home</a>
            <a class="item" href="#">Platos Típicos</a>
            <a class="item" href="#">Platos Bandera</a>
            <a class="item" href="#">Restaurantes</a>
            <a class="item" href="#">Nuestros Clientes</a>
            <a class="item" href="#">Contactenos</a>
        </nav>
        <header>
            <h1 id="h1cab">Gastronomía en el Perú</h1>
            
            <p id="pf">Fecha:</p>
            <p id="ph">Hora:</p>
        </header>
        <section>
            <h2 class="h2cab">Conociendo nuestra Gastronomía</h2>
            <article class="blok">
                
                <p class="pblok">
                    La historia culinaria de la comida peruana se remonta
                    desde las culturas preincas.
                    Teniendo ya ingredientes como el maíz, y las distintas
                    variedades
                    de papas y especias, nuestra cocina cambió con la
                    llegada de los conquistadores
                    españoles y las diferentes olas migratorias de
                    ciudadanos chinos, europeos, indios y
                    japoneses (principalmente a lo largo del siglo XIX). <br/>
                </p>
            </article>
            <article class="blok">
                
                <p class="pblok">
                    Los nativos peruanos ya habían domesticado unas 1000
                    variedades
                    de papas antes de que llegaran los españoles. Este
                    tubérculo es probablemente
                    la principal contribución de los incas al mundo entero,
                    sin mencionar que fue
                    fundamental para su dieta. Además de la papa, la cocina
                    inca también comprendía
                    cereales como la quinua y el maíz; carnes como la alpaca
                    y el cuy; frutas y
                    obviamente muchos tipos de pimientos picantes.
                </p>
            </article>
        </section>
    </div>

```

```

<article class="blok">
    
    <p class="pblok">
        Después de la independencia (1821), una ola consistente
        de inmigrantes europeos
        llegó al Perú y sus gastronomías, en particular la francesa
        e italiana,
        proporcionaron un giro adicional al crisol culinario.<br/>
        La verdadera revolución gastronómica llegó del Lejano
        Oriente.
        Los chinos conservaron fervientemente su identidad
        cultural y sus tradiciones,
        se establecieron en la “Calle Capón”, donde
        introdujeron nuevas técnicas de fritura e ingredientes
        como la salsa de soya o el quíon.
    </p>
</article>
<article class="blok">
    
    <p class="pblok">
        En la década de 1950, los japoneses habían erradicado
        este prejuicio.
        Sus restaurantes servían deliciosos platos de pescado y
        mariscos que pocos
        podrían resistir. De hecho, fue su toque culinario el que
        recreó el Ceviche y
        el Tiradito tal como los conocemos hoy en día.
    </p>
</article>
</section>
<footer>
    
</footer>
</div>
</body>

```

### Definiendo el archivo estilos\_1.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```

body{
    background-color: gray;
    margin: 0px auto;
}

#principal{
    width: 80%;
    float: left;
    margin: 0 10% 0 10%;
    border-radius: 20px 20px 0 0;
    background-color: white;
}

```

```
header,section,footer{  
    width: 100%;  
    float: left;  
    margin-bottom: 5px;  
}  
nav{  
    width: 90%;  
    margin:0 5% 0 5%;  
    float: left;  
    text-align:right;  
}  
.item{  
    width:15%;  
    margin:10px 0 10px 0;  
    height:20px;  
    float:left;  
    color:gray;  
    text-align:center;  
    text-decoration:none;  
}  
.item:hover{  
    color:blue;  
    text-decoration:underline;  
}  
#h1cab{  
    text-align: center;  
    color:gray;  
}  
#imgcab{  
    width: 100%;  
    height: 250px;  
    float: left;  
    margin:0;  
    opacity: 1;  
}  
#pf, #ph {  
    width: 48%;  
    height: 20px;  
    float: left;  
    border-radius: 10px;  
    color: white;  
    background-color: gray;  
    padding-left:2%;  
}  
.h2cab {  
    text-align: center;  
    color: gray;  
}  
.blok{  
    width: 22%;  
    min-height: 200px;  
    border:1px solid;
```

```

float: left;
margin: 1%;
color: gray;
}

.imgblok{
width: 98%;
height: 250px;
margin: 3px 1% 3px 1%;
float: left;
}

.pblok{
text-align: justify;
font-size: 16px;
margin-left: 2px;
}

#imgpie{
width: 100%;
height: 70px;
float: left;
margin-top: 20px;
margin-bottom: 20px;
}

```

### Definiendo la programación.

1. En el archivo de JavaScript archivo\_1.js, implementa las funciones parpadear(), fade() y fadeOut() para realizar las animaciones a los elementos. Para ejecutar las funciones, las mencionamos al inicio del archivo, tal como se muestra.

```

pagina1.html JS archivo_1.js ...
js > JS archivo_1.js > ...
1 parpadear();
2 fade();
3
4 function parpadear()
5 {
6     let r=Math.floor(Math.random()*256);
7     let g=Math.floor(Math.random()*256);
8     let b=Math.floor(Math.random()*256);
9
10    document.getElementById("h1cab").style.color="rgb(" + r + "," + g + "," + b + ")";
11    setTimeout("parpadear()",1000);
12 }
13
14 function fade(){
15     document.getElementById("imgcab").style.opacity="0";
16     document.getElementById("imgcab").style.transition="all 1s ease";
17     setTimeout("fadeOut()",2000);
18 }
19
20 function fadeOut(){
21     document.getElementById("imgcab").style.opacity="1.0";
22     document.getElementById("imgcab").style.transition="all 1s ease";
23     setTimeout("fade()",2000);
24 }

```

Figura 74 : Caso práctico

Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina1.html, y presiona la tecla F5.

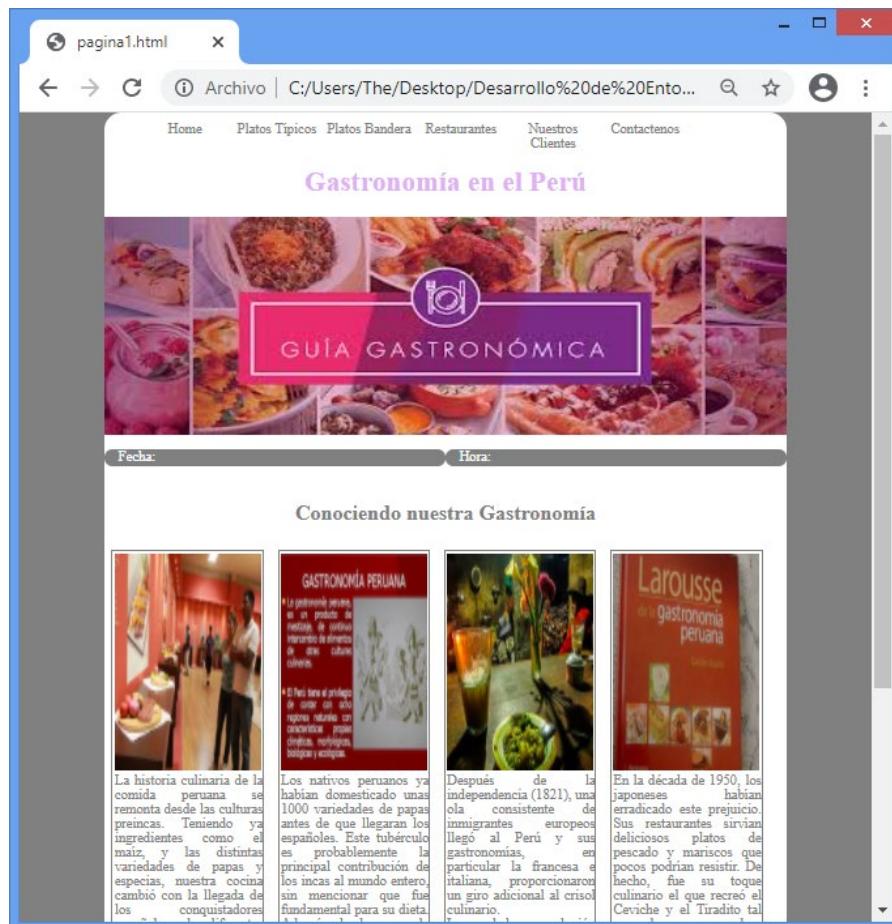


Figura 75: Caso práctico

Fuente.- Elaboración Propia

## LABORATORIO 2

### Trabajando con funciones

Implementa funciones para dar animaciones a los elementos de la cabecera de una página web:

- En el título `<h1>` de la cabecera de la página debe rolear o rotar las letras del título por intervalo de tiempo.
- En la imagen `<img>` de la cabecera de la página, debe implementar un carrusel de imágenes las cuales cambiarán por cada segundo
- En el bloque de id “`pf`”, visualizo la fecha del sistema
- En el bloque de id “`ph`” visualizo la hora del sistema la cual cambiará por cada segundo.

#### 1. Diseñando la página web

En esta etapa vamos a diseñar la página2.html, tal como se muestra

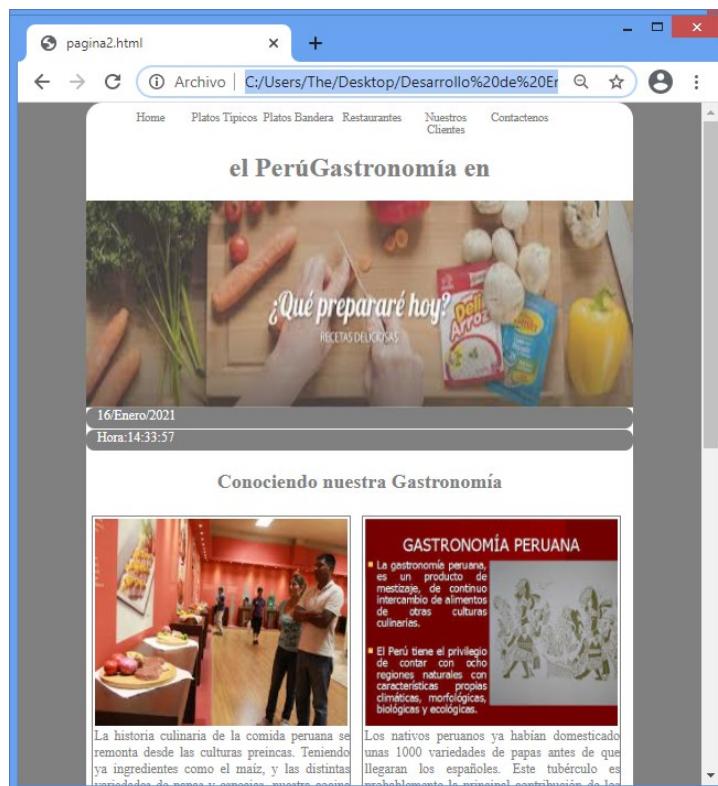


Figura 76 : Caso práctico

Fuente.- Elaboración Propia

En la página2.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_2.css y al archivo JS: archivo\_2.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5      <meta charset="utf-8">
6      <link href="css/estilos_1.css" rel="stylesheet" />
7  </head>
8  <body>
9      <div id="principal">
10         <nav> ...
11         </nav>
12         <header> ...
13         </header>
14         <section> ...
15         </section>
16         <footer>
17             
18         </footer>
19     </div>
20 </body>
21 </html>
22 <script src="js/archivo_2.js"></script>
23
24
25
26
27

```

Ln 24, Col 18 Tab Size: 4 UTF-8 with BOM CRLF HTML ⚙ Q

Figura 77: Caso práctico  
Fuente.- Elaboración Propia

## Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```

<body>
    <div id="principal">
        <nav>
            <a class="item" href="#">Home</a>
            <a class="item" href="#">Platos Típicos</a>
            <a class="item" href="#">Platos Bandera</a>
            <a class="item" href="#">Restaurantes</a>
            <a class="item" href="#">Nuestros Clientes</a>
            <a class="item" href="#">Contactenos</a>
        </nav>
        <header>
            <h1 id="h1cab">Gastronomía en el Perú</h1>
            
            <p id="pf">Fecha:</p>
            <p id="ph">Hora:</p>
        </header>
    </div>

```

```
<section>
    <h2 class="h2cab">Conociendo nuestra Gastronomía</h2>
    <article class="blok">
        
        <p class="pblok">
            La historia culinaria de la comida peruana se remonta desde las culturas preincas. Teniendo ya ingredientes como el maíz, y las distintas variedades de papas y especias, nuestra cocina cambió con la llegada de los conquistadores españoles y las diferentes olas migratorias de ciudadanos chinos, europeos, indios y japoneses (principalmente a lo largo del siglo XIX). <br/>
        </p>
    </article>
    <article class="blok">
        
        <p class="pblok">
            Los nativos peruanos ya habían domesticado unas 1000 variedades de papas antes de que llegaran los españoles. Este tubérculo es probablemente la principal contribución de los incas al mundo entero, sin mencionar que fue fundamental para su dieta. Además de la papa, la cocina inca también comprendía cereales como la quinua y el maíz; carnes como la alpaca y el cuy; frutas y obviamente muchos tipos de pimientos picantes.
        </p>
    </article>
    <article class="blok">
        
        <p class="pblok">
            Después de la independencia (1821), una ola consistente de inmigrantes europeos llegó al Perú y sus gastronomías, en particular la francesa e italiana, proporcionaron un giro adicional al crisol culinario.<br/> La verdadera revolución gastronómica llegó del Lejano Oriente. Los chinos conservaron fervientemente su identidad cultural y sus tradiciones, se establecieron en la "Calle Capón", donde introdujeron nuevas técnicas de fritura e ingredientes como la salsa de soya o el quión.
        </p>
    </article>
    <article class="blok">
        
    </article>
```

```

        <p class="pblok">
            En la década de 1950, los japoneses habían erradicado
            este prejuicio.
            Sus restaurantes servían deliciosos platos de pescado y
            mariscos que pocos
            podrían resistir. De hecho, fue su toque culinario el que
            recreó el Ceviche y
            el Tiradito tal como los conocemos hoy en día.
        </p>
    </article>
</section>
<footer>
    
</footer>
</div>
</body>

```

### Definiendo el archivo estilos\_1.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```

body{
    background-color: gray;
    margin: 0px auto;
}
#principal{
    width: 80%;
    height: auto;
    float: left;
    margin: 0 10% 0 10%;
    border-radius: 20px 20px 0 0;
    background-color: white;
}
header, section, footer{
    width: 100%;
    height: auto;
    float: left;
    margin-bottom: 5px;
}
nav{
    width: 90%;
    margin: 0 5% 0 5%;
    height: auto;
    float: left;
    text-align: right;
}
.item{
    width: 15%;
    margin: 10px 0 10px 0;
    height: 20px;
    float: left;
}

```

```
        color:gray;
        text-align:center;
        text-decoration:none;
    }
.item:hover{
        color:blue;
        text-decoration:underline;
}
#h1cab{
        text-align: center;
        color:gray;
}
#imgcab{
        width: 100%;
        height: 250px;
        float: left;
        margin:0;
        opacity: 1;
}
#pf, #ph {
        width: 48%;
        height: 20px;
        float: left;
        border-radius: 10px;
        color: white;
        background-color: gray;
        padding-left:2%;
}
.h2cab {
        text-align: center;
        color: gray;
}
.blok{
        width: 22%;
        min-height: 200px;
        height: auto;
        border:1px solid;
        float: left;
        margin:1%;
        color: gray;
}
.imgblok{
        width:98%;
        height:250px;
        margin:3px 1% 3px 1%;
        float:left;
}
.pblok{
        text-align:justify;
        font-size:16px;
        margin-left:2px;
}
```

```
#imgpie{
    width: 100%;
    height: 70px;
    float: left;
    margin-top:20px;
    margin-bottom: 20px;
}
```

### Definiendo la programación.

1. Defina la función carrusel() donde oculta la imagen en un segundo (transition), luego ejecuta la función roleo al cabo de 1 segundo.
2. Defina la función roleo(), donde hace cambio de imagen y se visualiza al cabo de 1 segundo, luego ejecuta la función carrusel() al cabo de 1 segundo.

The screenshot shows a code editor window with the tab 'JS archivo\_2.js' selected. The code is as follows:

```

1 //definiendo variables globales
2 var x=1;
3
4 function carrusel(){
5     x+=1;
6     if(x>5) {x=1;}
7
8     document.getElementById("imgcab").style.opacity='0';
9     document.getElementById("imgcab").style.transition="all 1s ease";
10
11    setTimeout("roleo()",1000);
12 }
13
14 function roleo(){
15     document.getElementById("imgcab").setAttribute("src","img/ban"+x+".jpg");
16
17     document.getElementById("imgcab").style.opacity='1';
18     document.getElementById("imgcab").style.transition="all 1s ease";
19     setTimeout("carrusel()",2000);
20 }

```

Annotations with red arrows point to the 'carrusel()' and 'roleo()' functions. The annotation for 'carrusel()' contains the text: 'funcion carrusel() desvanece la imagen en un segundo'. The annotation for 'roleo()' contains the text: 'funcion roleo(), realiza cambio de imagen y se muestra en un segundo'. The status bar at the bottom of the editor shows: Ln 59, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ 🌐

Figura 78: Caso práctico  
Fuente.- Elaboración Propia

3. Defina la función reloj(), la cual imprime en el selector de id “ph” la hora del sistema, repitiendo la operación al cabo de 1 segundo, uso del setTimeout.

```

pagina2.html JS archivo_2.js ...
js > JS archivo_2.js > ...
1 //definiendo variables globales
2 var x=1;
3
4 > function carrusel(){...}
12 }
13
14 > function roleo(){...}
20 }
21
22 function reloj(){
23     let f=new Date();
24     let h=f.getHours();
25     let m=f.getMinutes()<10 ? "0"+f.getMinutes() : f.getMinutes();
26     let s=f.getSeconds()<10 ? "0"+f.getSeconds() : f.getSeconds();
27     document.getElementById("ph").innerHTML="Hora:"+h+":"+m+":"+s;
28     setTimeout("reloj()",1000);
29 }

```

función reloj(), donde imprime la hora del sistema y ejecuta un setTimeout, para ejecutar nuevamente la función

Figura 79: Caso práctico  
Fuente.- Elaboración Propia

4. Defina la función fecha, donde imprime la fecha del sistema en el selector de id “pf”. En esta función, el mes estará expresado en letras.

```

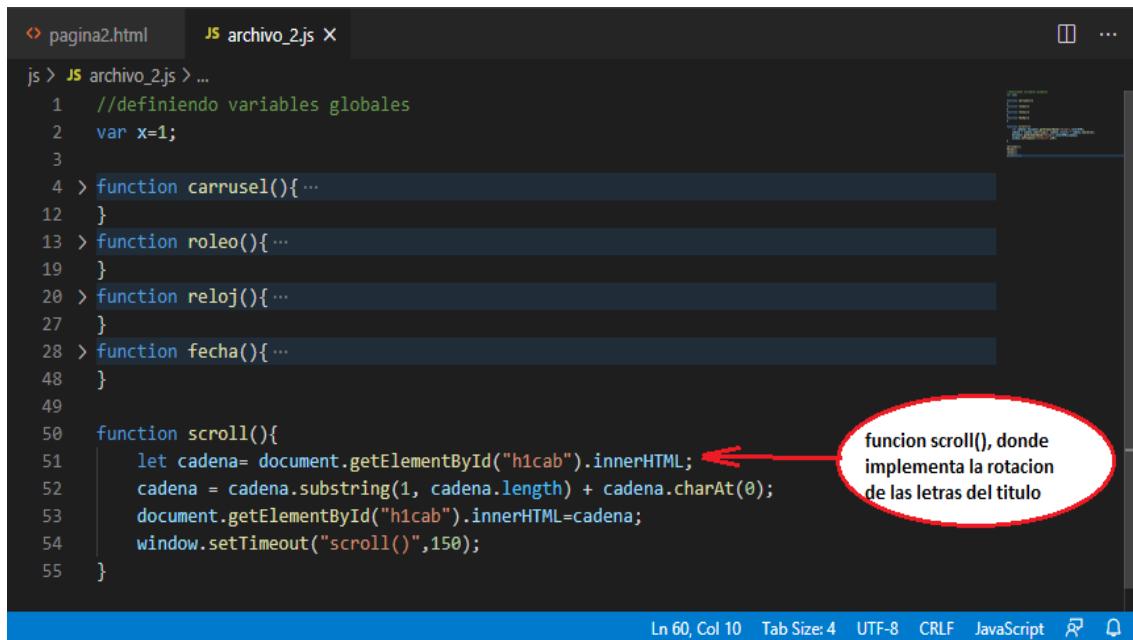
pagina2.html JS archivo_2.js ...
js > JS archivo_2.js > ...
1 //definiendo variables globales
2 var x=1;
3
4 > function carrusel(){...}
12 }
13 > function roleo(){...}
19 }
20 > function reloj(){...}
27 }
28
29 function fecha(){
30     var f=new Date();
31     var dia=f.getDate();
32     var y=f.getFullYear();
33     var mes;
34     switch(f.getMonth()){
35         case 0: mes="Enero";break;
36         case 1: mes="Febrero";break;
37         case 2: mes="Marzo";break;
38         case 3: mes="Abril";break;
39         case 4: mes="Mayo";break;
40         case 5: mes="Junio";break;
41         case 6: mes="Julio";break;
42         case 7: mes="Agosto";break;
43         case 8: mes="Setiembre";break;
44         case 9: mes="Octubre";break;
45         case 10: mes="Noviembre";break;
46         default: mes="Diciembre";break;
47     }
48     document.getElementById("pf").innerText=dia+"/"+mes+"/"+y;
49 }

```

función fecha(), donde imprime la fecha del sistema, el cual imprime el mes en letras.

Figura 80 : Caso práctico  
Fuente.- Elaboración Propia

5. Defina la función scroll(), donde las letras del título de la página estarán rotando por cada intervalo de tiempo.



```

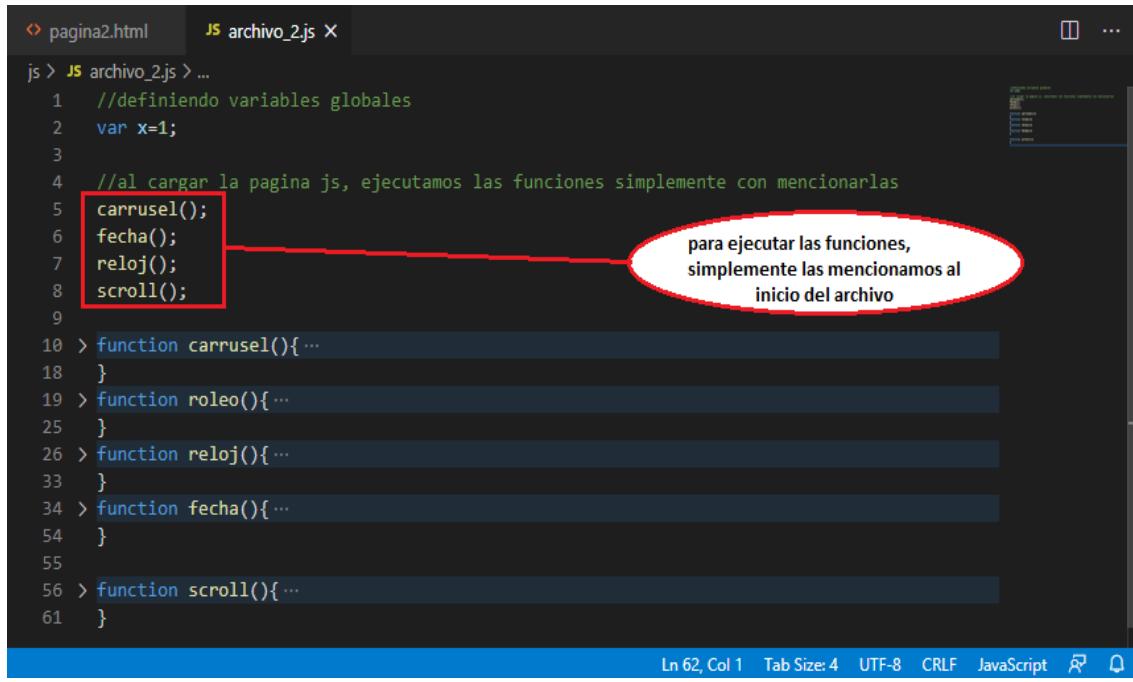
pagina2.html JS archivo_2.js ...
js > JS archivo_2.js > ...
1 //definiendo variables globales
2 var x=1;
3
4 > function carrusel(){...}
12 }
13 > function roleo(){...}
19 }
20 > function reloj(){...}
27 }
28 > function fecha(){...}
48 }
49
50 function scroll(){
51   let cadena= document.getElementById("h1cab").innerHTML;
52   cadena = cadena.substring(1, cadena.length) + cadena.charAt(0);
53   document.getElementById("h1cab").innerHTML=cadena;
54   window.setTimeout("scroll()",150);
55 }

```

Ln 60, Col 10 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ 🔍

Figura 81 : Caso práctico  
Fuente.- Elaboración Propia

6. Para ejecutar las funciones, las invocamos al inicio del archivo, tal como se muestra.



```

pagina2.html JS archivo_2.js ...
js > JS archivo_2.js > ...
1 //definiendo variables globales
2 var x=1;
3
4 //al cargar la pagina js, ejecutamos las funciones simplemente con mencionarlas
5 carrusel();
6 fecha();
7 reloj();
8 scroll();
9
10 > function carrusel(){...}
18 }
19 > function roleo(){...}
25 }
26 > function reloj(){...}
33 }
34 > function fecha(){...}
54 }
55
56 > function scroll(){...
61 }

```

Ln 62, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ 🔍

Figura 82: Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina2.html, y presiona la tecla F5.

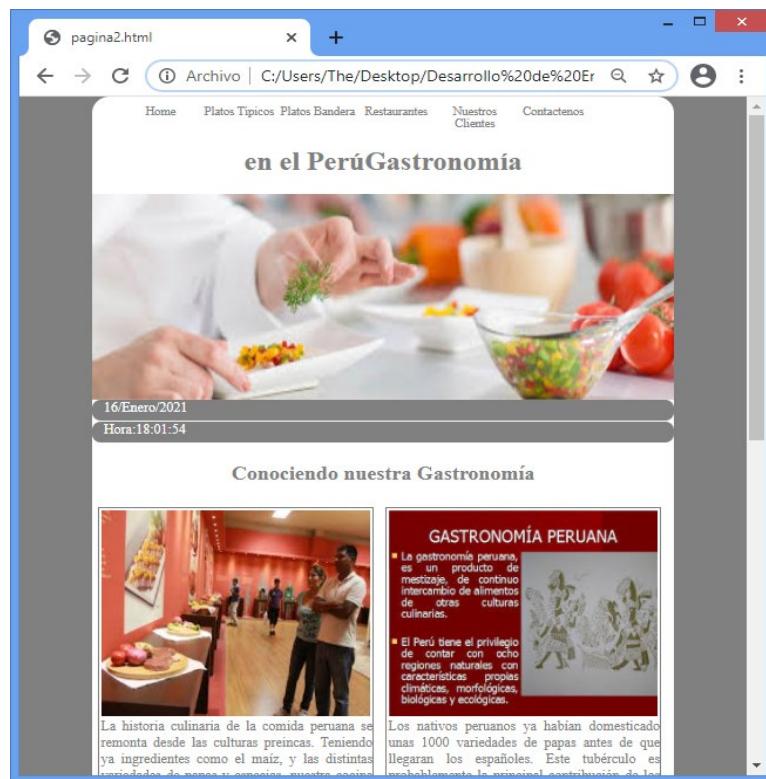


Figura 83 : Caso práctico

Fuente.- Elaboración Propia

# Resumen

1. Una función no es más que un bloque de enunciados que componen un comportamiento que puede ser invocado las veces que sea necesario.
2. Para ejecutar la función posteriormente no hay más que invocar su nombre en cualquier momento y desde cualquier parte de un código, con una excepción: la función debe haber sido definida anteriormente.
3. Una función puede recibir argumentos o parámetros, que se especifican entre los paréntesis que van tras el nombre de la función, y se separan por comas.
4. Si se define una variable dentro de una función, esa variable sólo existe para esa función, y otras funciones no pueden acceder a su valor a menos que lo reciban como un argumento o parámetro.
5. Una función con return es un módulo de programa que puede recibir datos de entrada a través de variables locales denominadas argumentos y que retorna un resultado al punto donde es invocado.
6. Los eventos cronometrados que nos proporciona el objeto Window es: setInterval() y setTimeout(), los cuales ejecutan un método por cada intervalo de tiempo expresados en milisegundos.

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://www.desarrolloweb.com/articulos/583.php>
- <http://www.desarrolloweb.com/articulos/584.php>
- <http://www.desarrolloweb.com/articulos/585.php>
- [http://librosweb.es/libro/JavaScript/capitulo\\_4/funciones.html](http://librosweb.es/libro/JavaScript/capitulo_4/funciones.html)

## 3.2. EVENTOS

### 3.2.1. Introducción

En JavaScript, una forma de controlar las acciones de los visitantes y definir un determinado comportamiento que se produzcan en la página, es a través de los eventos. Cuando un usuario visita una página web e interactúa con ella se desencadenan los eventos y con JavaScript podemos definir qué queremos que ocurra cuando se produzcan.

Los eventos son acciones u ocurrencias que suceden en el sistema que está programando y que el sistema le informa para que pueda responder de alguna manera si lo desea. Por ejemplo, si el usuario hace clic en un botón en una página web, es posible que desee responder a esa acción mostrando un cuadro de información.

Para desarrollar páginas interactivas, debemos definir eventos, porque con ellos podemos responder a las acciones de los usuarios.

Cada evento disponible tiene un **controlador de eventos**, que es un bloque de código (generalmente una función JavaScript definida por el usuario) que se ejecutará cuando se active el evento.

Cuando dicho bloque de código se define para ejecutarse en respuesta a la ejecución del evento, decimos que estamos registrando un controlador de eventos.

Tenga en cuenta que los controladores de eventos a veces se llaman oyentes de eventos, son bastante intercambiables para nuestros propósitos, aunque estrictamente hablando, trabajan juntos. El oyente escucha si ocurre el evento y el controlador es el código que se ejecuta en respuesta a que ocurra.

En el siguiente ejemplo, tenemos un solo <button>, que cuando se presiona, visualiza el cuadro del número ingresado como parámetro:

```
<button>Cuadro del número</button>
El JavaScript se ve así:

const btn = document.querySelector('button');

function cuadrado(number) {
    var x=number*number;
    alert("el cuadrado de " + number + " es " + x);
}
```

btn.onclick = cuadrado(5);

#### Cómo se define un evento

Para definir las acciones que queremos realizar al producirse un evento utilizamos los manejadores de eventos. El manejador de eventos se coloca en la etiqueta HTML, como un atributo, del elemento de la página que queremos que responda a las acciones del usuario.

Por ejemplo, tenemos el manejador de eventos **onclick**, que sirve para describir acciones que queremos que se ejecuten cuando se hace un click. Si queremos que al hacer **click** sobre un botón ejecute una operación, escribimos el manejador onclick en la etiqueta <INPUT type=button> de ese botón. Algo parecido a esto:

```
<INPUT type=button value="pulsame" onclick="sentencias_JavaScript...">
```

Se coloca un **atributo** nuevo en la etiqueta que tiene el mismo nombre que el evento, en este caso onclick. El atributo se iguala a las sentencias JavaScript que queremos que se ejecuten al producirse el evento.

### 3.2.2. Tipos de eventos

En este modelo, cada elemento o etiqueta XHTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos XHTML diferentes y un mismo elemento XHTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo on, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina onclick y el evento asociado a la acción de mover el ratón se denomina onmousemove.

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>

Evento	Descripción	Elementos para los que está definido
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>

Evento	Descripción	Elementos para los que está definido
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo, al cerrar el navegador)	<body>

### 3.2.3. Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede responder ante cualquier evento que se produzca durante su ejecución.

Las funciones JavaScript que se definen para cada evento se denominan "manejador de eventos" y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos XHTML.
- Manejadores como funciones JavaScript externas.
- Manejadores "semánticos".

#### Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez menos profesional de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un **atributo** del propio elemento XHTML.

En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Hola Mundo');" />
```

#### Manejadores de eventos y variable this

JavaScript define una variable especial llamada **this** que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable **this** para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del <div>, el color del borde se muestra de color negro. Cuando el ratón sale del <div>, se vuelve a mostrar el borde con el color gris claro original.

Elemento <div> original:

```
<div id="div1" style="width:150px; height:60px; border:thin solid silver">  
  Sección de contenidos...  
</div>
```

Si no se utiliza la variable `this`, el código necesario para modificar el color de los bordes sería el siguiente:

```
<div id="div1" onmouseover="document.getElementById('div1').style.borderColor='black';"  
  onmouseout="document.getElementById('div1').style.borderColor='silver';">  
  Sección de contenidos...  
</div>
```

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable `this`, que hace referencia al elemento XHTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:

```
<div id="div1" onmouseover="this.style.borderColor='black';"  
  onmouseout="this.style.borderColor='silver';">  
  Sección de contenidos...  
</div>
```

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente, aunque se modifique el valor del atributo `id` del `<div>`.

### Manejadores de eventos como funciones externas

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

```
function muestraMensaje() {  
  alert('Gracias por pinchar');  
}  
  
<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el **atributo** del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento. El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable `this` y, por tanto, es necesario pasar esta variable como parámetro a la función:

```
function resalta(elemento) {  
  switch(elemento.style.borderColor) {  
    case 'silver':  
      case 'silver silver silver silver':
```

```

case '#c0c0c0': elemento.style.borderColor = 'black'; break;
case 'black':
case 'black black black black':
case '#000000': elemento.style.borderColor = 'silver'; break;
}
}

<div style="width:150px; height:60px; border:thin solid silver" onmouseover="resalta(this)"
onmouseout="resalta(this)">
  Sección de contenidos...
</div>

```

En el ejemplo anterior, la función externa es llamada con el parámetro this, que dentro de la función se denomina elemento. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad borderColor.

Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor black, Internet Explorer lo almacena como black black black black y Opera almacena su representación hexadecimal #000000.

### **Manejadores de eventos semánticos**

Los métodos que se han visto para añadir manejadores de eventos (como atributos XHTML y como funciones externas) tienen un grave inconveniente: "ensucian" el código XHTML de la página.

Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (XHTML) y su aspecto o presentación (CSS). Siempre que sea posible, también se recomienda separar los contenidos (XHTML) y su comportamiento o programación (JavaScript).

Mezclar el código JavaScript con los elementos XHTML solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

```
<input id="btn1" type="button" value="Haz Click" onclick="alert('Gracias');" />
```

Se puede transformar en:

```

// Función externa
function muestraMensaje() {
  alert('Gracias por pinchar');
}

// Asignar la función externa al elemento
document.getElementById("btn1").onclick = muestraMensaje;

```

```
//Asignar la función utilizando setAttribute  
document.getElementById("btn1").setAttribute("onclick",muestraMensaje());
```

La gran ventaja de este método es que el código XHTML resultante es muy "limpio", ya que no se mezcla con el código JavaScript. Además, dentro de las funciones externas asignadas sí que se puede utilizar la variable this para referirse al elemento que provoca el evento.

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento onload:

```
window.onload = function() {  
    document.getElementById("pinchable").onclick = muestraMensaje;  
}
```

### 3.2.4. Manejo de listener

En el desarrollo de una aplicación, todo programador busca implementar su desarrollo que lo acerque lo más posible al ideal de separar las capas de contenido, presentación y comportamiento de un documento. Al implementar el atributo onClick deberíamos definir a la capa que pertenece propiamente al comportamiento.

Al implementar el comportamiento de los nodos del documento, se define el método **addEventListener** (inglés): nos sirve para registrar un evento a un objeto en específico. El objeto específico puede ser un simple elemento en un archivo, el mismo documento, una ventana o un XMLHttpRequest.

**addEventListener ()** es una función o método incorporado en JavaScript que toma el evento y lo pone a escuchar, y un segundo argumento para llamar cada vez que se desencadena el evento descrito. Se puede agregar cualquier número de controladores de eventos a un solo elemento sin sobrescribir los controladores de eventos existentes.

La sintaxis de addEventListener es muy sencilla:

**elemento.addEventListener('evento',función,booleano);**

Donde:

elemento: es cualquier elemento presente en un documento, al que accedemos por el medio que elijamos, bien por su id, por su etiqueta o las propiedades de otro nodo.

evento: es el suceso ocurrido sobre el elemento, con o sin interacción del usuario, como vimos en la sección de sintaxis de JavaScript relativa a los eventos.

función: es cualquier función definida que queramos que se ejecute cuando ocurra el evento.

booleano es un valor que define el orden del flujo de eventos.

```
<!DOCTYPE html>  
<html>  
<head>  
<title>DOM Event Example</title>
```

```
<style>
#t { border: 1px solid red }
#t1 { background-color: pink; }
</style>

<script>
// función para cambiar el contenido t2
function modifyText() {
    var t2 = document.getElementById("t2");
    t2.firstChild.nodeValue = "three";
}

// función para agregar un evento a t
function load() {
    var el = document.getElementById("t");
    el.addEventListener("click", modifyText, false);
}

document.addEventListener("DOMContentLoaded", load, false);
</script>

</head>
<body>

<table id="t">
    <tr><td id="t1">one</td></tr>
    <tr><td id="t2">two</td></tr>
</table>

</body>
</html>
```

En el ejemplo anterior, `modifyText()` es un listener para los eventos click registrados utilizando `addEventListener()`. Un click en cualquier parte de la tabla notificara al handler y ejecutara la función `modifyText()`.

Sus beneficios son los siguientes:

- Permite agregar más de un listener a un solo evento. Esto es particularmente útil para las librerías DHTML o las Extensiones de Mozilla que deben funcionar bien, incluso si se utilizan otras librerías/extensiones.
- Da un control más detallado de la fase en la que el listener se activa (capturing vs. bubbling)
- Funciona en cualquier elemento del DOM, no únicamente con elementos de HTML.

## LABORATORIO 1

### Trabajando con Eventos

Implementa los eventos de la página web (load y resize) para:

- Cargar la fecha y hora
- Ejecutar el carrusel de imágenes
- Al rediseñar el tamaño de la página, los elementos de esta deben ser reubicados.

### 1. Creando el Sitio Web

Defina una carpeta llamado SitioWeb\_04, en ella crea las siguientes subcarpetas, tal como se muestra.

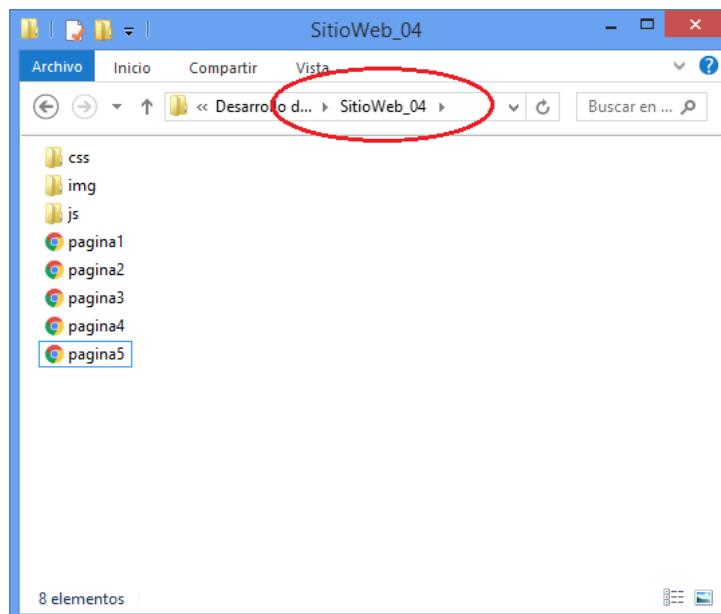


Figura 84 : Caso práctico  
Fuente.- Elaboración Propia

### 2. Trabajando con Visual Studio Code

Para crear una página web, abrir el aplicativo Visual Studio Code.

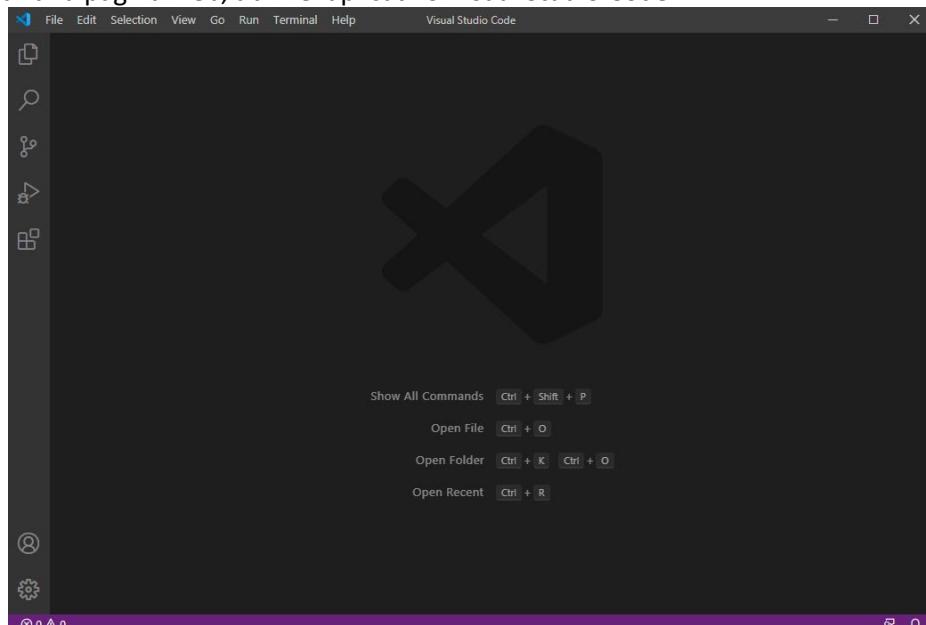


Figura 85 : Caso práctico  
Fuente.- Elaboración Propia

Para trabajar con la carpeta llamada SitioWeb\_04 en Visual Studio Code, arrastre la carpeta hacia la ventana del Visual, tal como se muestra; y luego soltarla.

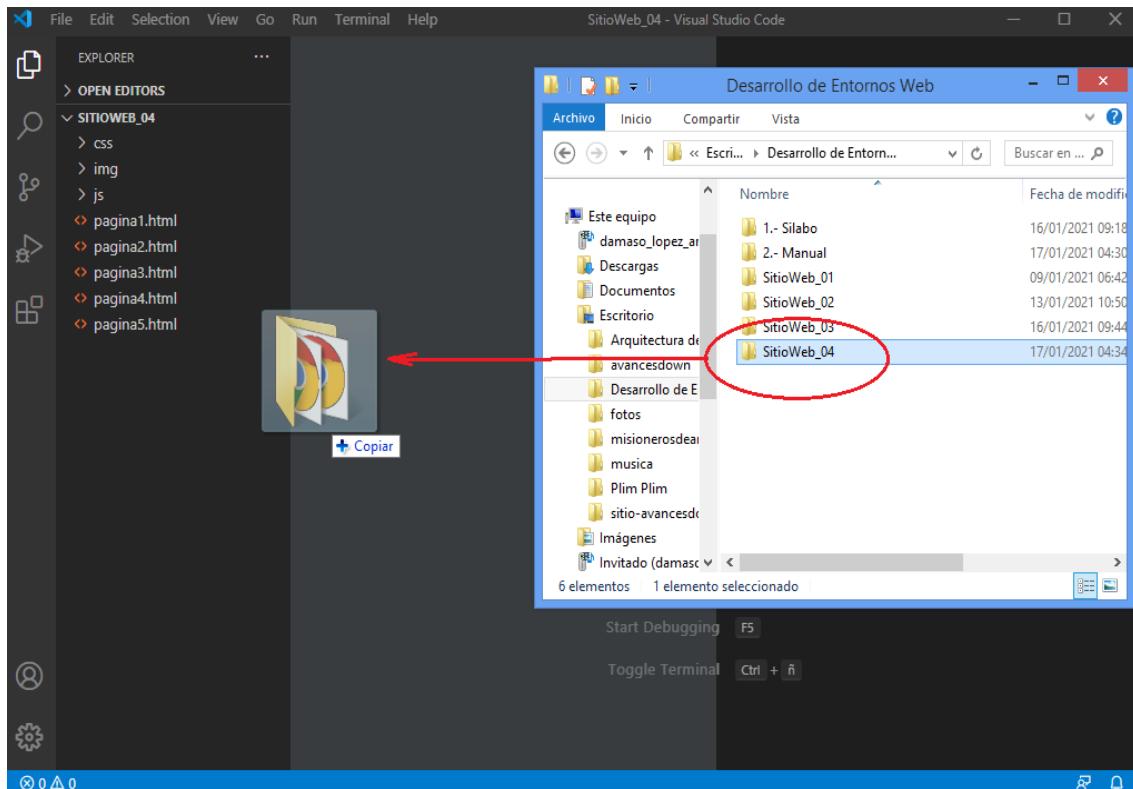


Figura 86: Caso práctico  
Fuente.- Elaboración Propia

Donde se visualiza la carpeta en la parte izquierda de la ventana del aplicativo

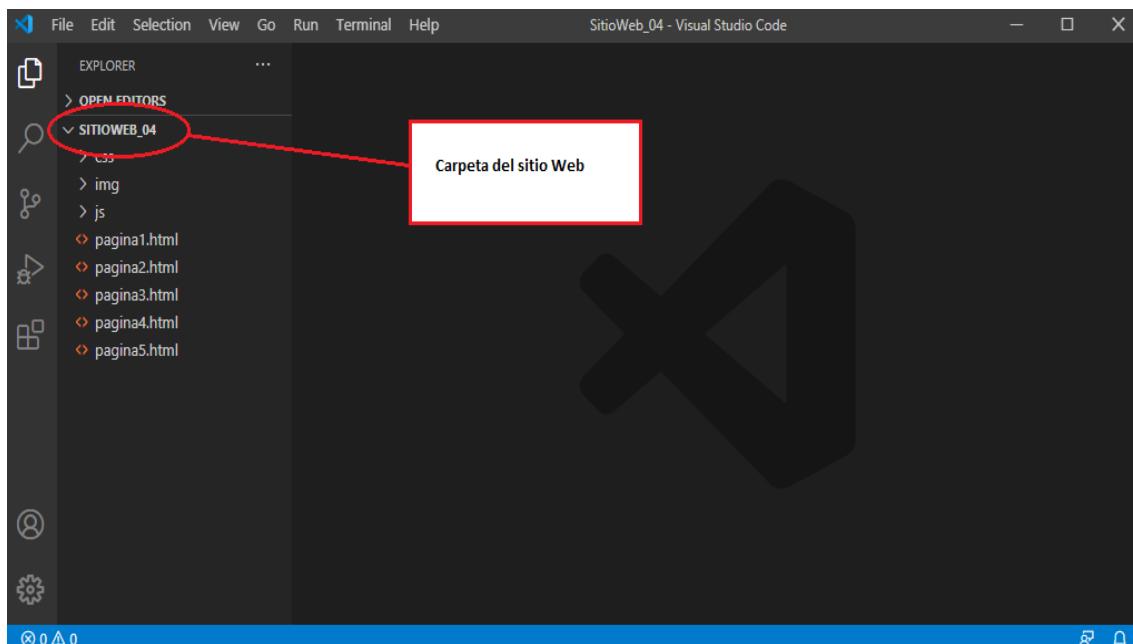


Figura 87 : Caso práctico  
Fuente.- Elaboración Propia

## 1. Diseñando la página web

En esta etapa vamos a diseñar la página1.html, tal como se muestra

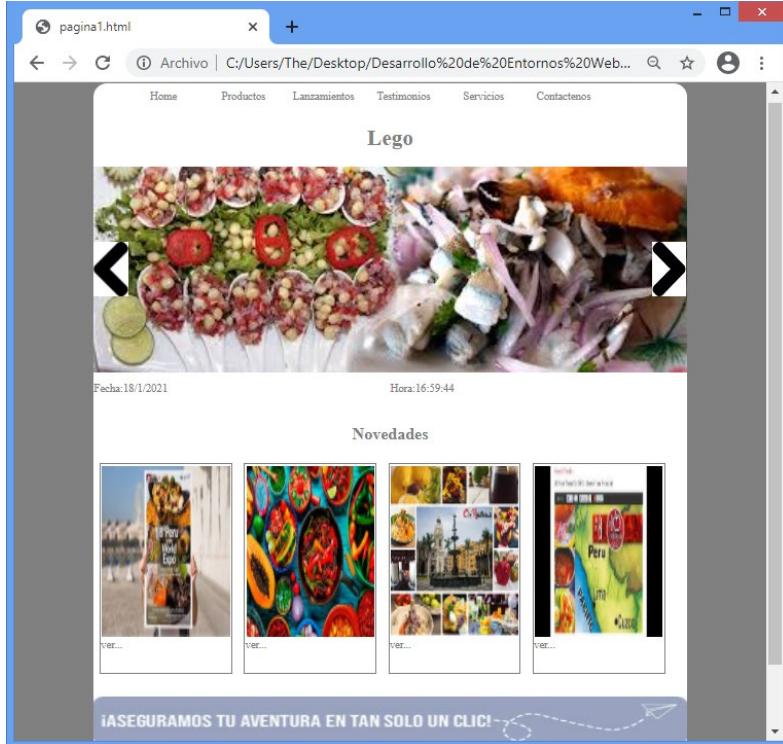


Figura 88: Caso práctico  
Fuente.- Elaboración Propia

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_1.css y al archivo JS: archivo\_1.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <link href="css/estilos_1.css" rel="stylesheet" />
6   </head>
7   <body>
8     <div id="principal">
9       <nav> ...
10      </nav>
11      <header> ...
12      </header>
13      <section> ...
14      </section>
15      <footer>
16        
17      </footer>
18    </div>
19  </body>
20 </html>
21 <script src="js/archivo_1.js"></script>
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75

```

Enlazando al archivo estilos\_1.css

Enlazando al archivo de JavaScript: archivo\_1.js

Figura 89: Caso práctico  
Fuente.- Elaboración Propia

### Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```
<body>
    <div id="principal">
        <nav>
            <a class="item" href="#">Home
            <a class="item" href="pagina2.html">Productos
            <a class="item" href="#">Lanzamientos
            <a class="item" href="#">Testimonios
            <a class="item" href="#">Servicios
            <a class="item" href="#">Contactenos
        </nav>
        <header>
            <h1 id="h1cab">Lego</h1>
            <div id="divcab">
                
                
                
            </div>
            <p id="pf">Que dia es?</p>
            <p id="ph">Que hora es?</p>
        </header>
        <section>
            <h2 class="h2cab">Novedades</h2>
            <article class="blok">
                
                <p class="ptitulo">ver...</p>
                <p class="pblok">
                    Una fiesta, feria o festival gastronómico o de comida es
                    un evento de ocio cuyo tema central
                    son los alimentos y/o bebidas, bien sea sobre una
                    técnica culinaria o producto en particular
                    o sobre la gastronomía de una región, una denominación
                    de origen, entre otros.
                </p>
            </article>
            <article class="blok">
                
                <p class="ptitulo">ver...</p>
                <p class="pblok">
                    Gracias a los accidentes geográficos que posee la sierra
                    del Perú, la gastronomía
                    andina se caracteriza por llevar a la mesa gran variedad
                    de alimentos nutritivos como son
                    la papa, el camote, las habas, el maíz, entre otros.
                </p>
            </article>
            <article class="blok">
                
                <p class="ptitulo">ver...</p>
                <p class="pblok">
                    Perú fue reconocido el 28 de noviembre como el mejor
                    destino culinario del mundo
                </p>
            </article>
        </section>
    </div>
</body>
```

También fue premiado como por octavo año consecutivo en los World Travel Awards. Picchu como la mayor atracción mejor destino cultural y se hizo lo propio con Machu turística a nivel global.

```

        </p>
    </article>
    <article class="blok">
        
        <p class="ptitulo">ver...</p>
        <p class="pblok">
            El sitio web especializado en el mercado gastronómico,
            The Food Channel, ubicó a la comida
            peruana en el puesto ocho de sus 10 primeras
            tendencias para el próximo año.
            El ranking incluye, aparte de cocina internacional,
            movimientos culturales y
            comportamientos que influirán en el consumo
            gastronómico.
        </p>
    </article>
</section>
<footer>
    
</footer>
</div>
</body>
```

### Definiendo el archivo estilos\_1.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```

body{
    background-color: gray;
}
#principal{
    width: 80%;
    height: auto;
    float: left;
    margin: 0 10% 0 10%;
    border-radius: 20px;
    background-color: white;
}

section, footer{
    width: 100%;
    float: left;
    margin-bottom: 5px;
}
header{
    width: 100%;
```

```
float: left;
margin-bottom: 5px;
overflow: hidden;
}
#divcab{
    width: 100%;
    height: auto;
    position: relative;
    float: left;
}
#izq,#der{
    position: absolute;
    width: 50px;
    height: 80px;
    left:0px;
    top:110px;
    z-index: 999;
}
nav{
    width: 90%;
    margin:0 5% 0 5%;
    height: auto;
    float: left;
    text-align:right;
}
.item{
    width:15%;
    margin:10px 0 10px 0;
    height:20px;
    float:left;
    color:gray;
    text-align:center;
    text-decoration:none;
}
.item:hover{
    color:blue;
    text-decoration:underline;
}
#h1cab{
    text-align: center;
    color:gray;
}
#imgcab{
    width: 100%;
    height: 300px;
    float: left;
    margin:0;
}
#pf, #ph {
    width: 50%;
    height: 20px;
```

```
float: left;
border-radius: 10px;
color: gray;
}
.h2cab {
    text-align: center;
    color: gray;
}
#ph2{
    text-align:justify;
    color:gray;
    margin-right:49%;
    margin-left:1%;
    float:left;
}
.blok{
    width: 22%;
    min-height: 200px;
    height: auto;
    border:1px solid;
    float: left;
    margin:1%;
    color: gray;
}
}
.imgblok{
    width:98%;
    height:250px;
    margin:3px 1% 3px 1%;
    float:left;
}
.pblok{
    text-align:justify;
    font-size:16px;
    margin-left:2px;
    visibility:hidden;
    height:0px;
}
}
#imgpie{
    width: 100%;
    height: 70px;
    float: left;
    margin-top:20px;
    margin-bottom: 20px;
}
```

### Definiendo la programación.

- En el archivo de JavaScript archivo\_1.js, implementa las funciones carrusel() y cambio() para realizar las animaciones de la imagen de la cabecera de la página.



```

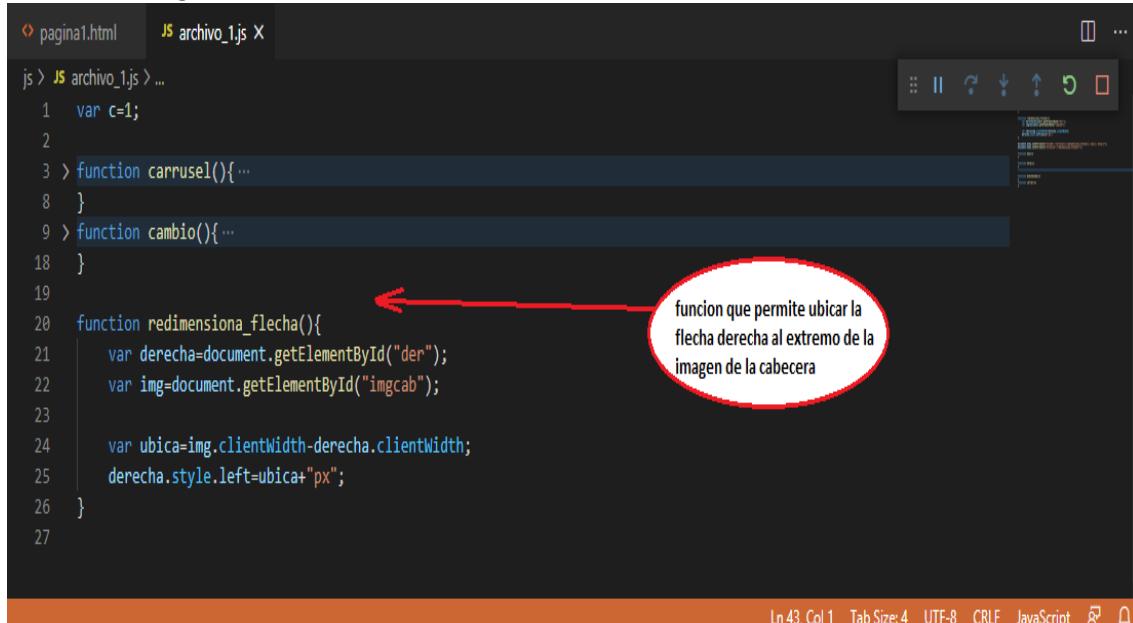
pagina1.html JS archivo_1.js ...
js > JS archivo_1.js > ...
1 var c=1;
2
3 function carrusel(){
4     document.querySelector("#imgcab").style.opacity="0";
5     document.querySelector("#imgcab").style.transition="all 1s";
6
7     setTimeout("cambio()",1000);
8 }
9 function cambio(){
10    c++;
11    if(c>5){c=1;}
12    document.querySelector("#imgcab").setAttribute("src","img/ban"+c+".jpg");
13
14    document.querySelector("#imgcab").style.opacity="1";
15    document.querySelector("#imgcab").style.transition="all 1s";
16
17    setTimeout("carrusel()",2000);
18 }
19

```

Ln 43, Col 1 Tab Size:4 UTF-8 CRLF JavaScript ⚡ ⚡

Figura 90 : Caso práctico  
Fuente.- Elaboración Propia

- Defina la función redimensiona\_flecha(), donde ubica la flecha derecha al extremo de la imagen de la cabecera



```

pagina1.html JS archivo_1.js ...
js > JS archivo_1.js > ...
1 var c=1;
2
3 > function carrusel(){ ... }
4
5 > function cambio(){ ... }
6
7
8 function redimensiona_flecha(){
9     var derecha=document.getElementById("der");
10    var img=document.getElementById("imgcab");
11
12    var ubica=img.clientWidth-derecha.clientWidth;
13    derecha.style.left=ubica+"px";
14
15 }
16
17
18
19
20
21
22
23
24
25
26
27

```

Ln 43, Col 1 Tab Size:4 UTF-8 CRLF JavaScript ⚡ ⚡

Figura 91 : Caso práctico  
Fuente.- Elaboración Propia

3. Defina la función dia(), la cual imprime la fecha del sistema
4. Defina la función hora(), donde imprime la hora, minutos y segundos por cada intervalo de 1 segundo (uso del setTimeout)

```

pagina1.html JS archivo_1.js
js > JS archivo_1.js > hora
1 var c=1;
2
3 > function carrusel(){ ... }
4
5 > function cambio(){ ... }
6
7
8 > function redimensiona_flecha(){ ... }
9
10
11
12
13
14 > function dia(){
15     var f=new Date();
16     document.querySelector("#pf").innerText="Fecha:"+f.getDate() "/" +
17         (f.getMonth()+1) "/" +f.getFullYear();
18 }
19
20
21
22
23
24 > function hora(){
25     var f=new Date();
26     document.querySelector("#ph").innerText="Hora:" +f.getHours() ":" +
27         f.getMinutes() ":" +f.getSeconds();
28     setTimeout("hora()",1000);
29 }
30
31
32
33
34
35
36
37
38
39
40

```

función dia(), donde imprime la fecha del sistema

función hora(), donde imprime la hora, minutos y segundos por cada intervalo de 1 segundo

Figura 92 : Caso práctico

Fuente.- Elaboración Propia

5. Utilizando el método setAttribute, asignaremos al evento load(), los métodos indicados; de igual modo, asignamos al evento resize(), el método redimensiona\_flecha.

```

pagina1.html JS archivo_1.js
js > JS archivo_1.js > cerrar
1 var c=1;
2
3 > function carrusel(){ ... }
4
5 > function cambio(){ ... }
6
7
8 > function redimensiona_flecha(){ ... }
9
10
11
12
13
14 > function dia(){ ... }
15
16
17
18 > function hora(){ ... }
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34 > function hora(){
35 }
36
37
38
39
40
41 document.body.setAttribute("onload","carrusel();redimensiona_flecha();dia();hora()");
42
43 document.body.setAttribute("onresize","redimensiona_flecha()");
44

```

utilizando setAttribute, asignar al evento load, los métodos: carrusel(), redimensiona\_flecha, dia(), hora()

utilizando setAttribute, asignar al evento resize el metodo redimensiona\_flecha

Figura 93 : Caso práctico

Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina1.html, y presiona la tecla F5.

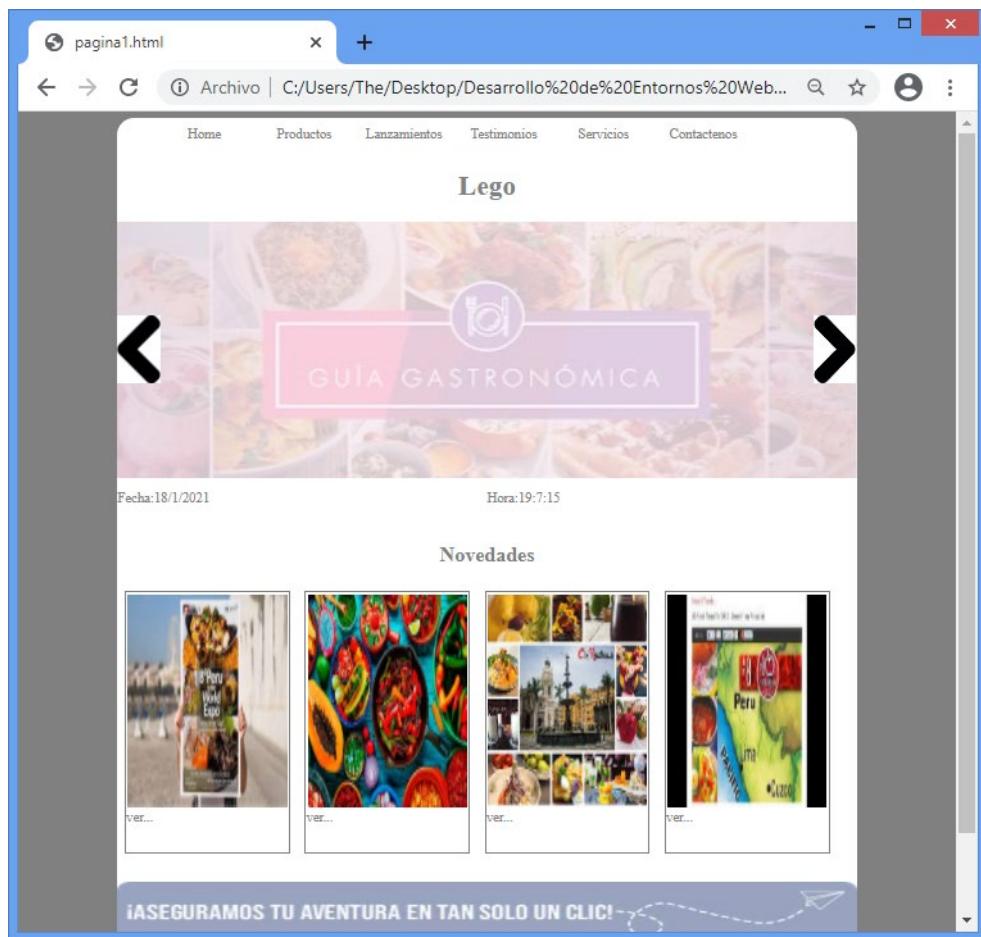


Figura 94: Caso práctico  
Fuente.- Elaboración Propia

## LABORATORIO 2

### Implementando el evento Click

Implementa el evento Click a los siguientes elementos de la página

- Programa los elementos flecha izquierda y flecha derecha para retroceder o avanzar las imágenes de la cabecera
- Programa los elementos de class: ítem cocina, donde al hacer Clic visualizamos la imagen de la gastronomía seleccionada.
- Programa los elementos de la class título, donde al hacer Clic, ocultamos la imagen y visualizamos el texto; y al hacer click, visualizar nuevamente la imagen.

#### 1. Diseñando la página web

En esta etapa vamos a diseñar la página2.html, tal como se muestra

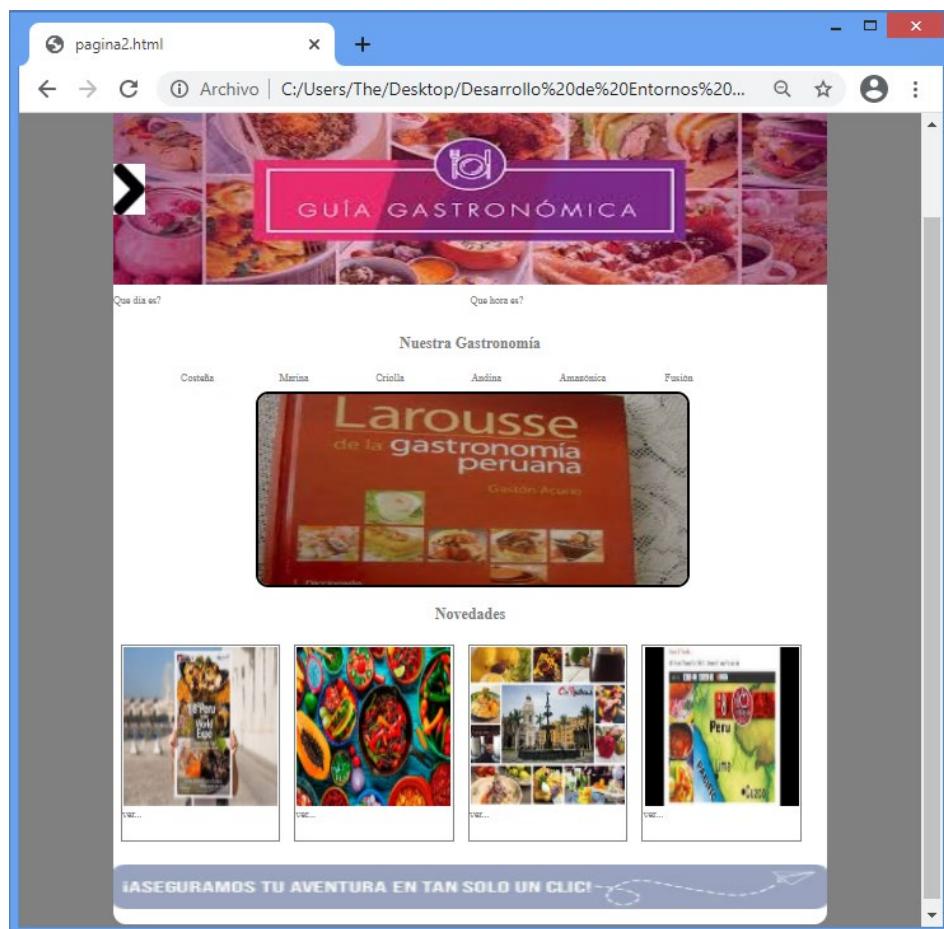
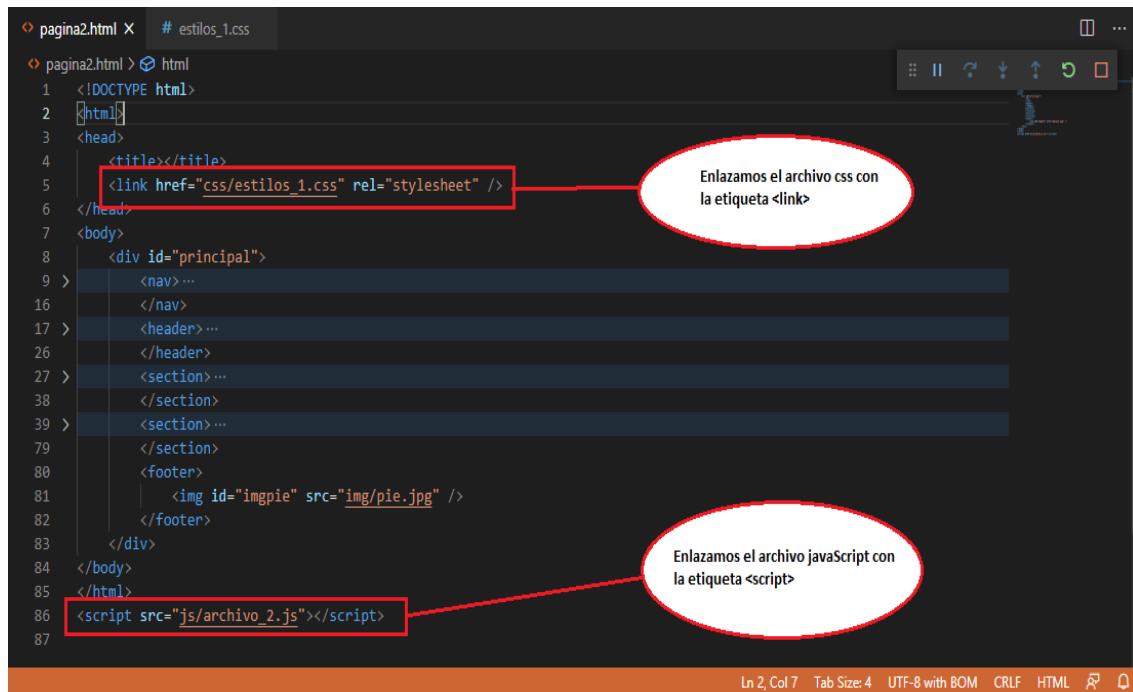


Figura 95 : Caso práctico  
Fuente.- Elaboración Propia

En la página2.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_1.css y al archivo JS: archivo\_2.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.



```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <link href="css/estilos_1.css" rel="stylesheet" />
</head>
<body>
    <div id="principal">
        <nav>...
        </nav>
        <header>...
        </header>
        <section>...
        </section>
        <section>...
        </section>
        <footer>
            
        </footer>
    </div>
</body>
</html>
<script src="js/archivo_2.js"></script>

```

Figura 96: Caso práctico  
Fuente.- Elaboración Propia

### Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```

<body>
    <div id="principal">
        <nav>
            <a class="item" href="#">Home</a>
            <a class="item" href="pagina2.html">Productos</a>
            <a class="item" href="#">Lanzamientos</a>
            <a class="item" href="#">Testimonios</a>
            <a class="item" href="#">Servicios</a>
            <a class="item" href="#">Contactenos</a>
        </nav>
        <header>
            <h1 id="h1cab">Viajando a través de la Gastronomía</h1>
            <div id="divcab">
                <!--flecha izquierda-->
                
                
                <!--flecha derecha-->
                
            </div>
            <p id="pf">Que dia es?</p>
            <p id="ph">Que hora es?</p>
        </header>
        <section>

```

```
<nav>
<h2 class="h2cab">Nuestra Gastronomía</h2>
<!--elemento <a> de class itemcocina, almacena un valor en data-
cocina-->
<a class="itemcocina" href="#" data-cocina="costeña.jpg">Costeña</a>
<a class="itemcocina" href="#" data-cocina="marina.jpg">Marina</a>
<a class="itemcocina" href="#" data-cocina="criolla.jpg">Criolla</a>
<a class="itemcocina" href="#" data-cocina="andina.jpg">Andina</a>
<a class="itemcocina" href="#" data-
cocina="amazonica.jpg">Amazónica</a>
<a class="itemcocina" href="#" data-cocina="fusion.jpg">Fusión</a>
</nav>

</section>
<section>
    <h2 class="h2cab">Novedades</h2>
    <article class="blok">
        
        <p class="ptitulo">ver...</p>
        <p class="pblok">
            Una fiesta, feria o festival gastronómico o de comida es
            un evento de ocio cuyo tema central
            son los alimentos y/o bebidas, bien sea sobre una
            técnica culinaria o producto en particular
            o sobre la gastronomía de una región, una denominación
            de origen, entre otros.
        </p>
    </article>
    <article class="blok">
        
        <p class="ptitulo">ver...</p>
        <p class="pblok">
            Gracias a los accidentes geográficos que posee la sierra
            del Perú, la gastronomía
            andina se caracteriza por llevar a la mesa gran variedad
            de alimentos nutritivos como son
            la papa, el camote, las habas, el maíz, entre otros.
        </p>
    </article>
    <article class="blok">
        
        <p class="ptitulo">ver...</p>
        <p class="pblok">
            Perú fue reconocido el 28 de noviembre como el mejor
            destino culinario del mundo
            por octavo año consecutivo en los World Travel Awards.
        </p>
    </article>

```

También fue premiado como  
Picchu como la mayor atracción

mejor destino cultural y se hizo lo propio con Machu  
turística a nivel global.

```

<article class="blok">
    
    <p class="ptitulo">ver...</p>
    <p class="pblok">
        El sitio web especializado en el mercado gastronómico,
        The Food Channel, ubicó a la comida
        peruana en el puesto ocho de sus 10 primeras
        tendencias para el próximo año.
        El ranking incluye, aparte de cocina internacional,
        movimientos culturales y
        comportamientos que influirán en el consumo
        gastronómico.
    </p>
</article>
</section>
<footer>
    
</footer>
</div>
</body>

```

### Definiendo el archivo estilos\_2.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```

body{
    background-color: gray;
    margin:0px auto;
}
#principal{
    width: 80%;
    float: left;
    margin:0 10% 0 10%;
    border-radius: 20px 20px 0 0;
    background-color: white;
}
header,section,footer{
    width: 100%;
    float: left;
    margin-bottom: 5px;
}
nav{
    width: 90%;
    margin:0 5% 0 5%;
    float: left;
    text-align:right;
}
.item, .itemcocina{
    width:15%;
    margin:10px 0 10px 0;
    height:20px;
}

```

```
float:left;
color:gray;
text-align:center;
text-decoration:none;
}
.item:hover{
    color:blue;
    text-decoration:underline;
}
#h1cab{
    text-align: center;
    color:gray;
}
#imgcab{
    width: 100%;
    height: 250px;
    float: left;
    margin:0;
    opacity: 1;
}
#pf, #ph {
    width: 48%;
    height: 20px;
    float: left;
    border-radius: 10px;
    color: white;
    background-color: gray;
    padding-left:2%;
}
.h2cab {
    text-align: center;
    color: gray;
}
.blok{
    width: 22%;
    min-height: 200px;
    border:1px solid;
    float: left;
    margin:1%;
    color: gray;
}
.imgblok{
    width:98%;
    height:250px;
    margin:3px 1% 3px 1%;
    float:left;
}
.pblok{
    text-align:justify;
    font-size:16px;
    margin-left:2px;
}
```

```
#imgpie{
    width: 100%;
    height: 70px;
    float: left;
    margin-top:20px;
    margin-bottom: 20px;
}
```

### Definiendo la programación.

1. Defina la función redimensiona\_flecha() donde ubica la imagen flecha, al lado derecho de la imagen. Para registrar el evento resize asignar el método redimensiona\_flecha, utilice el método addEventListener, tal como se muestra.

The screenshot shows a code editor with two tabs: 'pagina2.html' and 'JS archivo\_2.js'. The 'JS archivo\_2.js' tab contains the following code:

```

1 var c=1;
2
3 function redimensiona_flecha(){
4     var derecha=document.getElementById("der");
5     var img=document.getElementById("imgcab");
6
7     var ubica=img.clientWidth-derecha.clientWidth;
8     derecha.style.left=ubica+"px";
9 }
10
11 > document.body.addEventListener("resize",redimensiona_flecha(),false);

```

Annotations with arrows point to specific parts of the code:

- An arrow points to the opening brace of the `redimensiona_flecha()` function with the text: "defina la función redimensiona\_flecha() donde ubica la flecha derecha".
- An arrow points to the `addEventListener` line with the text: "para registrar el evento resize al metodo utilice addEventListener".

At the bottom of the code editor, status bars show: Ln 3, Col 32 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ 🌐

Figura 97 : Caso práctico

Fuente.- Elaboración Propia

2. Defina la función hora(), la cual imprime en el selector de id "ph" la hora del sistema, repitiendo la operación al cabo de 1 segundo, uso del setTimeout. Para registrar el evento load del body, asignar el método hora() utilizando el método addEventListener.

The screenshot shows a code editor with two tabs: 'pagina2.html' and 'JS archivo\_2.js'. The 'JS archivo\_2.js' tab contains the following code:

```

1 var c=1;
2
3 > function redimensiona_flecha(){...}
9
10
11 function hora(){
12     let f=new Date();
13     let h=f.getHours();
14     let m=f.getMinutes()<10 ? "0"+f.getMinutes() : f.getMinutes();
15     let s=f.getSeconds()<10 ? "0"+f.getSeconds() : f.getSeconds();
16     document.querySelector("#ph").innerText="Hora:"+h+":"+m+":"+s;
17     setTimeout("hora()",1000);
18 }
19
20 document.body.addEventListener("resize",redimensiona_flecha(),false);
21
22 document.body.addEventListener("load",hora(),false);

```

Annotations with arrows point to specific parts of the code:

- An arrow points to the opening brace of the `hora()` function with the text: "funcion hora(), donde imprime la hora del sistema por cada segundo".
- An arrow points to the `document.body.addEventListener("load",hora(),false);` line with the text: "Registra el evento load del body al metodo hora()".

At the bottom of the code editor, status bars show: Ln 31, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ 🌐

Figura 98: Caso práctico

Fuente.- Elaboración Propia

3. Defina la función avanza() donde visualizamos la siguiente imagen de la cabecera; y la función retrocede() donde visualizamos la imagen anterior. Luego registra a cada elemento (der., izq.) el evento Clic asociando el método implementado, utilice addEventListener.

```

pagina2.html JS archivo_2.js
js > JS archivo_2.js > redimensiona_flecha
1 var c=1;
2
3 > function redimensiona_flecha(){...}
9 }
10 > function hora(){...}
17 }
18
19 function avanza(){
20     c++;
21     if(c>5)c=1;
22     document.getElementById("imgcab").setAttribute("src", "img/ban"+c+".jpg");
23 }
24
25 function retrocede(){
26     c--;
27     if(c<1)c=5;
28     document.getElementById("imgcab").setAttribute("src", "img/ban"+c+".jpg");
29 }
30
31 document.body.addEventListener("resize",redimensiona_flecha(),false);
32 document.body.addEventListener("load",hora(),false);
33
36 document.getElementById("izq").addEventListener("click",retrocede,false);
50
51 document.getElementById("der").addEventListener("click",avanza,false);

```

funcion avanza(), permite avanzar la imagen de la cabecera

funcion retrocede(), permite visualizar la imagen anterior de la cabecera

Registrar el evento click a cada elemento (flecha) con el método correspondiente

Figura 99: Caso práctico

Fuente.- Elaboración Propia

4. Defina la función novedades(), donde visualizamos la imagen gastronómica de acuerdo con el elemento seleccionamos. A continuación, a través de un for, registra a cada elemento item\_cocina, el evento Click asociando a la función novedades.

```

pagina2.html JS archivo_2.js
js > JS archivo_2.js > avanza
1 var c=1;
2
3 > function redimensiona_flecha(){...}
9 }
10 > function hora(){...}
17 }
18 > function avanza(){...}
22 }
23 > function retrocede(){...}
27 }
28
29 function novedades(i){
30     var data=document.querySelectorAll(".itemcocina")[i];
31     document.querySelector("#img_cocina").setAttribute("src","img/"+data.getAttribute("data-cocina"));
32 }
33
34 document.body.addEventListener("resize",redimensiona_flecha(),false);
35 document.body.addEventListener("load",hora(),false);
36 document.getElementById("izq").addEventListener("click",retrocede,false);
37 document.getElementById("der").addEventListener("click",avanza,false);
38
39 for(let x=0; x<document.getElementsByClassName("itemcocina").length; x++){
40     document.getElementsByClassName("itemcocina")[x].addEventListener("click",function(){novedades(x)},false);
41 }
42

```

funcion novedades() donde visualizamos la imagen del tipo de gastronomía

En cada elemento itemcocina, enlazamos el evento click con el metodo novedades[i]

Figura 100 : Caso práctico

Fuente.- Elaboración Propia

5. Defina la función mostrar(), donde visualizamos/ocultamos el texto el cual se encuentra oculta en el bloque article. A continuación, a través de un for, registra a cada elemento título, el evento Click asociando a la función mostrar

```

1  <!-- pagin2.html --> JS archivo_2.js <-->
2  js > JS archivo_2.js > ...
3
4  function mostrar(i){
5      var bloque=document.querySelectorAll(".pblok")[i];
6
7      if(bloque.clientHeight==0)
8      {
9          bloque.style.height="auto";
10         bloque.style.visibility="visible";
11         document.getElementsByClassName("ptitulo")[i].innerHTML="ocultar...";
12     }
13     else{
14         bloque.style.height="0px";
15         bloque.style.visibility="hidden";
16         document.getElementsByClassName("ptitulo")[i].innerHTML="ver...";
17     }
18 }
19
20 document.body.addEventListener("resize",redimensiona_flecha(),false);
21 document.body.addEventListener("load",hora(),false);
22 document.getElementById("izq").addEventListener("click",retrocede,false);
23 document.getElementById("der").addEventListener("click",avanza,false);
24
25 > for(let x=0; x<document.getElementsByClassName("itemcocina").length; x++){
26     }
27
28 for(let x=0; x<document.getElementsByClassName("ptitulo").length; x++){
29
30     document.getElementsByClassName("ptitulo")[x].addEventListener("click",function(){mostrar(x)},false);
31 }
32 
```

Ln 56, Col 2 Tab Size: 4 UTF-8 CRLF JavaScript ⌂ ⌂

Figura 101: Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina2.html, y presiona la tecla F5.

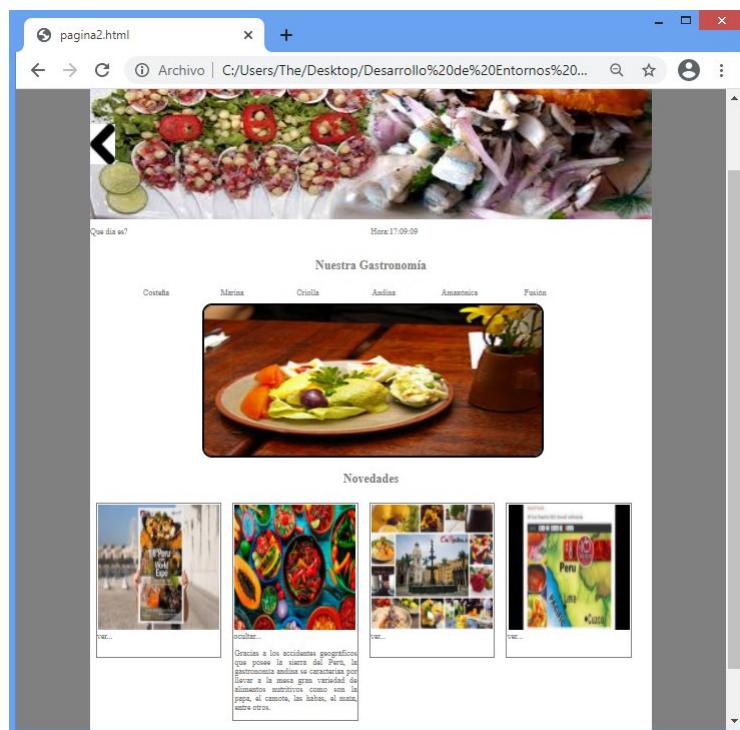


Figura 102 : Caso práctico  
Fuente.- Elaboración Propia

## LABORATORIO 3

### Implementando los eventos mouseOver, mouseOut

Implementa los eventos mouseOver y mouseOut a los siguientes elementos de la página

- Programa los elementos de las imágenes de redes sociales (imgsocial) donde al pasar el mouse (mouseover) cambio la imagen; y al salir cambiamos a la imagen original.
- Programa los elementos las imágenes de tradiciones y costumbres donde al pasar el mouse, cambiamos la apariencia de la imagen y al salir recupera su apariencia original.
- Programa los elementos de la class artblok, donde al pasar el mouse oculta la imagen y visualiza el texto y al salir visualizamos la imagen.

#### 1. Diseñando la página web

En esta etapa vamos a diseñar la página3.html, tal como se muestra

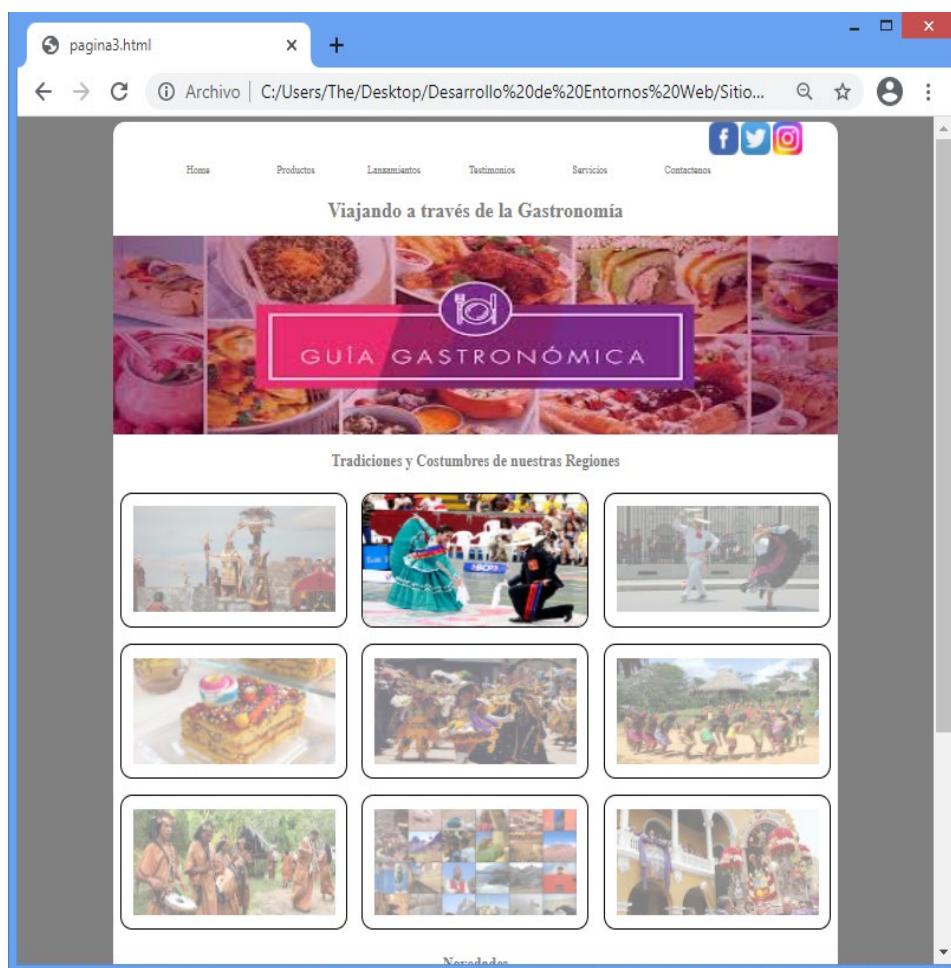


Figura :103 Caso práctico

Fuente.- Elaboración Propia

En la página3.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_3.css y al archivo JS: archivo\_3.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title></title>
5  |   <link href="css/estilos_3.css" rel="stylesheet" />
6  |</head>
7  <body>
8  |   <div id="principal">
9  |       <nav> ...
13 |       <nav> ...
14 |   </div>
21 |   <header> ...
22 |       
27 |   </header>
28 |   <section> ...
39 |   <section> ...
40 |       
76 |   </section>
77 |   <section>
78 |       
79 |   </section>
80 |   </div>
81 |</body>
82 |</html>
83 <script src="js/archivo_3.js"></script>

```

Figura 104: Caso práctico

Fuente.- Elaboración Propia

### Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```

<body>
    <div id="principal">
        <nav>
            
            
            
        </nav>
        <nav>
            <a class="item" href="#">Home</a>
            <a class="item" href="pagina2.html">Productos</a>
            <a class="item" href="#">Lanzamientos</a>
            <a class="item" href="#">Testimonios</a>
            <a class="item" href="#">Servicios</a>
            <a class="item" href="#">Contactenos</a>
        </nav>
        <header>
            <h1 id="h1cab">Viajando a través de la Gastronomía</h1>
            <div id="divcab">
                
            </div>
        </header>
        <section>

```

```
<h2 class="h2cab">Tradiciones y Costumbres de nuestras  
Regiones</h2>  
    <article class="art_trad"></article>  
        <article class="art_trad"></article>  
            <article class="art_trad"></article>  
                <article class="art_trad"></article>  
                    <article class="art_trad"></article>  
                        <article class="art_trad"></article>  
                            <article class="art_trad"></article>  
                                <article class="art_trad"></article>  
                                    <article class="art_trad"></article>  
            </section>  
            <section>  
                <h2 class="h2cab">Novedades</h2>  
                <article class="blok">  
                      
                    <p class="pblok">  
                        Una fiesta, feria o festival gastronómico o de comida es  
un evento de ocio cuyo tema central  
son los alimentos y/o bebidas, bien sea sobre una  
técnica culinaria o producto en particular  
o sobre la gastronomía de una región, una denominación  
de origen, entre otros.  
                </p>  
                </article>  
                <article class="blok">  
                      
                    <p class="pblok">  
                        Gracias a los accidentes geográficos que posee la sierra  
del Perú, la gastronomía  
andina se caracteriza por llevar a la mesa gran variedad  
de alimentos nutritivos como son  
la papa, el camote, las habas, el maíz, entre otros.  
                </p>  
                </article>  
                <article class="blok">  
                      
                    <p class="pblok">  
                        Perú fue reconocido el 28 de noviembre como el mejor  
destino culinario del mundo  
por octavo año consecutivo en los World Travel Awards.  
También fue premiado como

```

Picchu como la mayor atracción mejor destino cultural y se hizo lo propio con Machu turística a nivel global.

```

        </p>
    </article>
    <article class="blok">
        
        <p class="pblok">
            El sitio web especializado en el mercado gastronómico,
The Food Channel, ubicó a la comida peruana en el puesto ocho de sus 10 primeras tendencias para el próximo año. El ranking incluye, aparte de cocina internacional, movimientos culturales y comportamientos que influirán en el consumo gastronómico.
        </p>
    </article>
</section>
<footer>
    
</footer>
</div>
</body>
```

### Definiendo el archivo estilos\_3.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```

body{
    background-color: gray;
}
#principal{
    width: 80%;
    float: left;
    margin: 0 10% 0 10%;
    border-radius: 20px;
    background-color: white;
}
nav{
    width: 90%;
    margin: 0 5% 0 5%;
    float: left;
    text-align: right;
}
section, footer{
    width: 100%;
    float: left;
    margin-bottom: 5px;
}
header{
```

```
width: 100%;  
float: left;  
margin-bottom: 5px;  
overflow: hidden;  
}  
#divcab{  
    width: 100%;  
    height: auto;  
    position: relative;  
    float: left;  
}  
.imgsocial{  
    width: 50px;  
    height: 50px;  
    text-align: right;  
}  
.item{  
    width:15%;  
    margin:10px 0 10px 0;  
    height:20px;  
    float:left;  
    color:gray;  
    text-align:center;  
    text-decoration:none;  
}  
.item:hover{  
    color:blue;  
    text-decoration:underline;  
}  
#h1cab{  
    text-align: center;  
    color:gray;  
}  
  
#imgcab{  
    width: 100%;  
    height: 300px;  
    float: left;  
    margin:0;  
}  
.h2cab {  
    text-align: center;  
    color: gray;  
}  
.ph2{  
    text-align:justify;  
    color:gray;  
    margin-right:49%;  
    margin-left:1%;  
    float:left;  
}
```

```
.art_trad{  
    width: 31%;  
    height: 200px;  
    margin: 1%;  
    border:1px solid;  
    border-radius: 20px;  
    float: left;  
}  
.imgtradicion{  
    width: 90%;  
    height: 160px;  
    padding: 5% 20px 5% 20px;  
    opacity: 0.5;  
    float: left;  
}  
.blok{  
    width: 22%;  
    min-height: 200px;  
    border:1px solid;  
    float: left;  
    margin:1%;  
    color: gray;  
}  
.imgblok{  
    width:98%;  
    height:250px;  
    margin:3px 1% 3px 1%;  
    float:left;  
}  
.pblok{  
    text-align:justify;  
    font-size:16px;  
    margin-left:2px;  
    visibility:hidden;  
    height:0px;  
}  
#imgpie{  
    width: 100%;  
    height: 70px;  
    float: left;  
    margin-top:20px;  

```

### Definiendo la programación.

1. Defina la función **rscambio(n)**, el cual cambia la imagen de la red social al pasar el mouse. En este proceso utilizamos los métodos `indexOf()` y `substring()`. Del mismo modo, defina la función **rsoriginal(n)** el cual, al salir el mouse visualizamos la imagen original de la red social.

Para registrar los eventos mouseover y mouseout, utilizamos un for para asignar a cada elemento sus dos eventos, tal como se muestra.

```

pagina3.html JS archivo_3.js ...
js > JS archivo_3.js > ...
1 var c=1;
2
3 function rscambio(n){
4     let ubica=document.getElementsByClassName("imgsocial")[n].getAttribute("src");
5
6     let indice=ubica.indexOf(".");
7     let newubica=ubica.substring(0,indice)+"1"+ubica.substring(indice);
8
9     document.getElementsByClassName("imgsocial")[n].setAttribute("src",newubica);
10 }
11
12 function rsoriginal(n){
13     let ubica=document.getElementsByClassName("imgsocial")[n].getAttribute("src");
14
15     let indice=ubica.indexOf("1");
16     let newubica=ubica.substring(0,indice)+ubica.substring(indice+1);
17
18     document.getElementsByClassName("imgsocial")[n].setAttribute("src",newubica);
19 }
20
21 for(let i=0; i<document.getElementsByClassName("imgsocial").length; i++){
22     document.getElementsByClassName("imgsocial")[i].addEventListener("mouseover",function(){rscambio(i)},false);
23     document.getElementsByClassName("imgsocial")[i].addEventListener("mouseout",function(){rsoriginal(i)},false);
24 }

```

Figura 105 : Caso práctico  
Fuente.- Elaboración Propia

2. Defina la función **showImagen(n)**, la cual cambia las propiedades del style de imgtradicion al pasar el mouse. Defina la función **hideImagen(n)** la cual cambia las propiedades del style de imgtradicion. Para registrar los eventos mouseover y mouseout del elemento art\_tradicion, utilice un for, utilice el método addEventListener.

```

pagina3.html JS archivo_3.js ...
js > JS archivo_3.js > ...
24
25 function showImagen(n){
26     document.getElementsByClassName("imgtradicion")[n].setAttribute(
27         "style","width:100%; height:200px; padding:0; opacity:1; border-radius:20px; transition: all ease 1s");
28 }
29
30 function hideImagen(n){
31     document.getElementsByClassName("imgtradicion")[n].setAttribute(
32         "style","width:90%; height:160px; padding:5% 20px 5% 20px; opacity:0.5; transition: all ease 0.5s");
33 }
34
35 for(let x=0; x<document.getElementsByClassName("art_trad").length; x++){
36
37     document.getElementsByClassName("art_trad")[x].addEventListener("mouseover",function(){showImagen(x)},false);
38     document.getElementsByClassName("art_trad")[x].addEventListener("mouseout",function(){hideImagen(x)},false);
39 }
40

```

Figura 106: Caso práctico  
Fuente.- Elaboración Propia

3. Defina la función **mostrar(i)** donde visualizamos el texto y ocultamos la imagen; defina la función **ocultar(i)** donde visualizamos la imagen y ocultamos la imageb. Asociar los eventos

mouseover y mouseout de **blok** con las funciones previamente definidas. Utilice addEventListener.

```

pagina3.html JS archivo_3.js
js > JS archivo_3.js > ocultar
42
43 function mostrar(i){
44     var bloque=document.querySelectorAll(".pblok")[i];
45
46     if(bloque.clientHeight==0)
47     {
48         document.querySelectorAll(".imgblok")[i].setAttribute("style","visibility:hidden; height:0px");
49         bloque.style.height="auto";
50         bloque.style.visibility="visible";
51     }
52 }
53
54 function ocultar(i){
55     var bloque=document.querySelectorAll(".pblok")[i];
56
57     if(bloque.clientHeight>0)
58     [
59         bloque.style.height="0px";
60         bloque.style.visibility="hidden";
61         document.querySelectorAll(".imgblok")[i].setAttribute("style","visibility:visible; height:250px");
62     ]
63 }
64
65 for(let x=0; x<document.getElementsByClassName("blok").length; x++){
66     document.getElementsByClassName("blok")[x].addEventListener("mouseover",function(){mostrar(x)},false);
67     document.getElementsByClassName("blok")[x].addEventListener("mouseout",function(){ocultar(x)},false);
68 }
69

```

Figura 107: Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina3.html, y presiona la tecla F5.

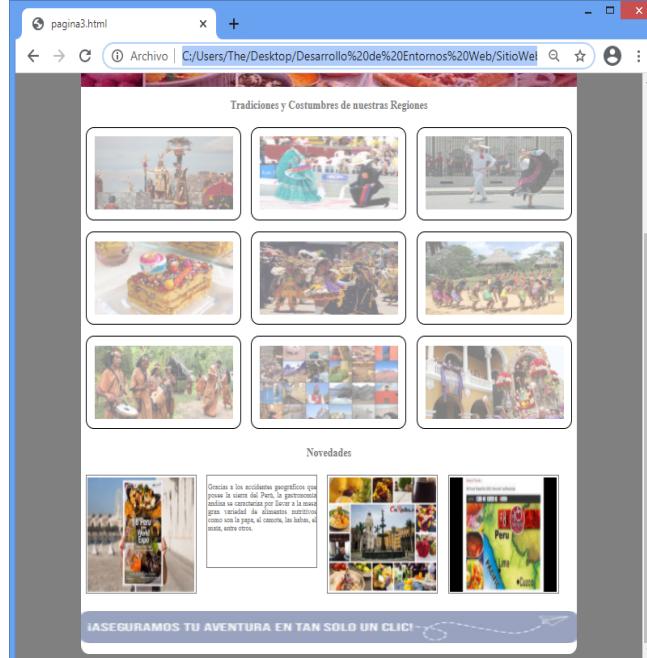


Figura 108: Caso práctico  
Fuente.- Elaboración Propia

# Resumen

1. Los eventos son acciones u ocurrencias que suceden en el sistema que está programando y que el sistema le informa para que pueda responder de alguna manera si lo desea.
2. Cada evento disponible tiene un controlador de eventos, que es un bloque de código (generalmente una función JavaScript definida por el usuario) que se ejecutará cuando se active el evento.
3. Para definir las acciones que queremos realizar al producirse un evento utilizamos los manejadores de eventos. El manejador de eventos se coloca en la etiqueta HTML, como un atributo, del elemento de la página que queremos que responda a las acciones del usuario.
4. En este modelo, cada elemento o etiqueta XHTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos XHTML diferentes y un mismo elemento XHTML puede tener asociados varios eventos diferentes.
5. Los manejadores de eventos como atributos XHTML, se trata del método más sencillo y a la vez menos profesional de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento XHTML.
6. Los manejadores de eventos como funciones externas, se realizan en aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.
7. Al implementar el comportamiento de los nodos del documento, se define el método addEventListener (inglés): nos sirve para registrar un evento a un objeto en específico. El objeto específico puede ser un simple elemento en un archivo, el mismo documento, una ventana o un XMLHttpRequest.

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://desarrolloweb.com/articulos/1235.php>
- <http://laplace.ucv.cl/Cursos/Old/FisComputacional/JavaScript/Tutorial/JavaScript5.html>
- <https://uniwebsidad.com/libros/JavaScript/capitulo-6/modelo-basico-de-eventos-2>
- [https://developer.mozilla.org/es/docs/Learn/JavaScript/Building\\_blocks/Eventos](https://developer.mozilla.org/es/docs/Learn/JavaScript/Building_blocks/Eventos)
- [https://developer.mozilla.org/es/docs/Web/Guide/DOM/Events/eventos\\_controlador](https://developer.mozilla.org/es/docs/Web/Guide/DOM/Events/eventos_controlador)
- <https://developer.mozilla.org/es/docs/Web/API/EventTarget/addEventListener>
- [http://www.codexempla.org/curso/curso\\_4\\_3\\_e.php](http://www.codexempla.org/curso/curso_4_3_e.php)



# MANEJO DE ARREGLOS

---

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno, con el lenguaje JavaScript, usando el modelo DOM, implementa un sitio web manejando arreglos de elementos y objetos.

## TEMARIO

- 4.1 Tema 5 : Arreglos**
  - 4..1.1 : Introducción
  - 4.1.2 : Definición de arreglos, creación
  - 4.1.3 : La clase array
  - 4.1.4 : Trabajando con arreglo de elementos
  
- 4.2 Tema 6 : ChildNodes**
  - 4.2.1 : Trabajando con childNodes
  - 4.2.2 : Métodos para acceder a nodos hijos
  - 4.2.3 : Métodos para agregar o quitar nodos

## ACTIVIDADES PROPUESTAS

- Mostrar la fecha del sistema.
- Mostrar un carrusel de imágenes.

## 4.1. ARREGLOS

### 4.1.1 Introducción

Los arreglos son una de las estructuras de datos más importantes en JavaScript.

En la figura podemos apreciar, un arreglo es una colección de elementos contiguos, más bien una estructura que nos permite almacenar secuencialmente datos, no necesariamente en orden. En JavaScript podemos almacenar, cualquier tipo de dato, y por lo general se almacenan datos del mismo tipo.

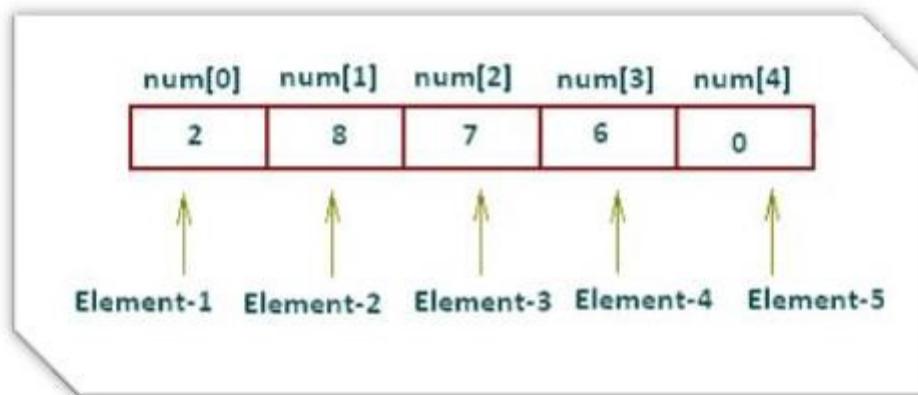


Figura 109: Arreglo

Fuente. - Tomado de <https://medium.com/@rodrwan/arreglos-en-JavaScript-4c9164e12a16>

### 4.1.2 Definición de arreglos, creación

En JavaScript un arreglo es una colección ordenada de elementos no homogéneos, cada elemento puede ser de un tipo de dato diferente. En JavaScript los arreglos empiezan con el índice 0.

Ejemplo: La siguiente sentencia genera un arreglo con 4 elementos de distinto tipo:

```
var b = [2.5, false, 6, "Hola"]
```

También es definido como un grupo o colección finita, no homogénea y ordenada de elementos. Un arreglo es un conjunto de datos o una estructura de datos no homogéneos que se encuentran ubicados en forma consecutiva en la memoria RAM.

#### Creación de arreglos

El primer paso para utilizar un arreglo es crearlo. Para ello utilizamos un objeto JavaScript ya implementado en el navegador. La sentencia para crear un objeto arreglo:

```
var miArray = [ ];
```

Esto crea un arreglo en la página que está ejecutándose. El array se crea sin ningún contenido, es decir, no tendrá ninguna casilla o compartimiento creado. También podemos crear el arreglo JavaScript especificando el número de compartimentos que va a tener e inicializándolo.

```
var miArray = ['5','6','7','8','9'];
```

En este caso indicamos que el arreglo va a tener 5 posiciones, donde cada casilla se le ha asignado un dato.

Los arreglos en JavaScript empiezan siempre en la posición 0, así que un array que tenga por ejemplo 5 posiciones tendrá casillas de la 0 a la 4. Para acceder a cada elemento del arreglo, utilizamos una estructura repetitiva

```
for (i=0;i<4;i++){  
    document.write("Posición " + i + " del array: " + miArray[i])  
    document.write("<br>")  
}
```

#### 4.1.3. La clase array

Los arrays son objetos similares a una lista cuyo prototipo proporciona métodos para efectuar operaciones de recorrido y de mutación. Tanto la longitud como el tipo de los elementos de un array son variables.

Dado que la longitud de un array puede cambiar en cualquier momento, y los datos se pueden almacenar en ubicaciones no contiguas, no hay garantía de que los arrays de JavaScript sean densos; esto depende de cómo el programador elija usarlos.

Es importante que nos fijemos que la palabra Array, en código JavaScript, se escribe con la primera letra en mayúscula. Como en JavaScript las mayúsculas y minúsculas sí que importan, si lo escribimos en minúscula no funcionará.

##### Definiendo un Array

Método 1º: definiendo y poblando simultáneamente

```
var tabla = new Array(25,35,12,34);
```

Método 2º: definiendo primero, poblando después cualquiera de las siguientes es válida en unos casos se especifica el número de elementos en otros no.

```
var tabla = new Array();  
var tabla = new Array(7);
```

Poblando el arreglo

```
tabla[0] = "lunes";  
tabla[1] = "martes";  
tabla[2] = "miércoles";  
tabla[3] = "jueves";
```

```
tabla[4] = "viernes";  
tabla[5] = "sábado";  
tabla[6] = "domingo";
```

Los arrays en JavaScript empiezan siempre en la posición 0, así que un array que tenga por ejemplo de 7 posiciones tendrá casillas de la 0 a la 6. Para recoger datos de un array lo hacemos igual: poniendo entre corchetes el índice de la posición a la que queremos acceder. Veamos cómo se imprimiría en la pantalla el contenido de un array.

```
for (var dia in tabla){  
    document.write("Dia de la semana: " + dia);  
    document.write("<br>")  
}
```

## Métodos de un Array

A continuación, se muestran los métodos de la clase Array y algunos ejemplos de cómo pueden mejorar nuestro código notablemente.

### 1. Agregar

push: Agrega al final del arreglo uno o más elementos. Devuelve la nueva longitud.

unshift: Agrega uno o más elementos al inicio. Devuelve la nueva longitud.

splice: Agrega y/o elimina elementos.

### 2. Eliminar

pop: Elimina el último elemento. Devuelve el elemento eliminado.

shift: Elimina el primer elemento. Devuelve el elemento eliminado.

### 3. Orden

reverse: Da vuelta el arreglo.

sort: Ordena los elementos.

### 4. Unión

concat: Une 2 arreglos.

join: Une los elementos en un string.

### 5. Posición

indexOf: Devuelve el índice del primer elemento encontrado.

lastIndexOf: Devuelve el índice del último elemento encontrado.

### 6. Recorrer

forEach: Ejecuta una función para cada elemento del arreglo.

#### Ejemplos

```
var arr = ['durazno', 'pera', 'manzana', 'banana', 'mandarina']
```

```
arr.pop()
```

Resultado: "mandarina" //el arreglo queda: ["durazno", "pera", "manzana", "banana"]

```
arr.shift()
```

Resultado: "durazno" //el arreglo queda: ["pera", "manzana", "banana"]

```
arr.push('naranja')
```

Resultado: //el arreglo queda: ["pera", "manzana", "banana", "naranja"]

```
arr.unshift('kiwi')
```

Resultado: 5 //el arreglo queda: ["kiwi", "pera", "manzana", "banana", "naranja"]

```
arr.splice(0,1)
```

Resultado: ["kiwi"] //el arreglo queda: ["pera", "manzana", "banana", "naranja"]

```
arr.splice(2,2)
```

Resultado: ["banana", "naranja"] //el arreglo queda: ["pera", "manzana"]

```
arr.splice(2,0, "banana", "naranja")
```

Resultado: [] //el arreglo queda: ["pera", "manzana", "banana", "naranja"]

```
arr.splice(0,1, "kiwi")
```

Resultado: ["pera"] //el arreglo queda: ["kiwi", "manzana", "banana", "naranja"]

```
arr.reverse()
```

Resultado: ["naranja", "banana", "manzana", "kiwi"] //el arreglo queda: ["naranja", "banana", "manzana", "kiwi"]

```
arr.sort()
```

Resultado: ["banana", "kiwi", "manzana", "naranja"] //el arreglo queda: ["banana", "kiwi", "manzana", "naranja"]

```
arr.concat(['pera', 'pomelo'])
```

Resultado: ["banana", "kiwi", "manzana", "naranja", "pera", "pomelo"] //el arreglo queda: ["banana", "kiwi", "manzana", "naranja"]

```
arr.join()
```

Resultado: "banana,kiwi,manzana,naranja"

```
arr.indexOf('naranja')
```

Resultado: 3

```
arr.indexOf('kiwi')
```

Resultado: 1

El siguiente código de JavaScript muestra los índices y los valores de cada elemento del arreglo:

```
<script>
    var días =
        ["lunes", "martes", "miercoles", "jueves", "viernes", "sabado", "domingo"];
    días.forEach(function(elemento,indice)
    {
        document.write(indice+" es "+lemento+"<br>");
    });
</script>
```

El resultado sería así:

0 es lunes  
1 es martes  
2 es miércoles  
3 es jueves  
4 es viernes  
5 es sábado  
6 es domingo

Otra forma de usar el código anterior sería así:

```
<script>
    function mostrarDias(elemento,indice){
        document.write(indice+" es "+elemento+"<br>");
    }
    var dias =
        ["lunes","martes","miercoles","jueves","viernes","sabado","domingo"];
    dias.forEach(mostrarDias);
</script>
```

Ejemplo 1: El siguiente código crea un arreglo de cadenas

```
<!DOCTYPE html>
<html lang="en">
<head>
<title></title>
<style>
    #miElemento{
        width:150px;
        height:250px;
    }
</style>
</head>
<body>
    <div id="miElemento">Texto del div</div>
</body>
<script>
    function aplicarEstilos(elemento,listaEstilos){
        for(var estilo in listaEstilos)
            elemento.style[estilo]=listaEstilos[estilo];
    }
    var elemento=document.getElementById('miElemento');
    var estilos={'border-top':'solid 1px red',
                'background':'#FF0',
                'font-size':'17px',
                '-webkit-transform':'rotate(90deg)',
                'margin':'40px 50px',
                'position':'absolute'};
    aplicarEstilos(elemento,estilos);
</script>
</html>
```

Ejemplo 2: El siguiente código crea una animación tipo slide de texto:

```
<!DOCTYPE html>
<html>
<head>
<style>
#wss{
    opacity:0;
    -webkit-transition:opacity 1.0s linear 0s;
    transition:opacity 1.0s linear 0s;
}
</style>
<script>
var wss_i = 0;
var wss_array = ["Cute","Happy",<u>Playful</u>,"Smart","Loyal"];
var wss_elem;
function wssNext(){
    wss_i++;
    wss_elem.style.opacity = 0;
    if(wss_i > (wss_array.length - 1)){
        wss_i = 0;
    }
    setTimeout('wssSlide()',1000);
}
function wssSlide(){
    wss_elem.innerHTML = wss_array[wss_i];
    wss_elem.style.opacity = 1;
    setTimeout('wssNext()',2000);
}
</script>
</head>
<body>
<h1>My dog is <span id="wss"></span></h1>
<script>wss_elem = document.getElementById("wss"); wssSlide(); </script>
</body>
</html>
```

#### 4.1.4. Trabajando con arreglo de elementos

JavaScript tiene acceso a todos los elementos de una página web. Si los objetos son del mismo tipo se agrupan en una colección. Por ejemplo, si en un documento hay cinco párrafos, se puede almacenar en una variable llamada párrafos, los cinco objetos p (de la etiqueta html que representa a los párrafos), de la siguiente manera:

```
var cantidad=document.getElementsByTagName("p");
```

Todo arreglo tiene una propiedad llamada length que devuelve la cantidad de elementos que contiene el arreglo. Por lo tanto, el siguiente código devolverá 5

```
alert("cantidad de parrafos: "+cantidad.length);
```

Así mismo, el siguiente código devuelve el texto del primer párrafo con la propiedad innerText:

```
alert("texto del parrafo 1: "+document.getElementsByTagName("p")[0].innerText);
```

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```
for(var i=0; i<parrafos.length; i++)
{
    var parrafo = parrafos[i];
}
```

La función getElementsByTagName() se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```
var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces = primerParrafo.getElementsByTagName("a");
```

En el siguiente código se evalúa la cantidad de párrafos y enlaces que hay en un documento html:

```
<script type="text/JavaScript">
window.onload = function()
{
    // Numero de enlaces de la pagina
    var enlaces = document.getElementsByTagName("a");

    // Direccion del penultimo enlace
    var penultimo = enlaces[enlaces.length-2];

    // Numero de enlaces que apuntan a http://prueba
    var contador = 0;

    for(var i=0; i<enlaces.length; i++)
    {
        // Es necesario comprobar los enlaces http://prueba y
        // http://prueba/ por las diferencias entre navegadores
        if(enlaces[i].getAttribute('href') == "http://prueba" || enlaces[i].getAttribute('href') ==
        "http://prueba/") {
            contador++;
        }
    }

    // Numero de enlaces del tercer párrafo
    var parrafos = document.getElementsByTagName("p");
    enlaces = parrafos[2].getElementsByTagName("a");
```

```
        alert("Numero de enlaces = "+enlaces.length+
          "\nEl penultimo enlace apunta a: "+penultimo.getAttribute('href')+
          "\n"+contador + " enlaces apuntan a http://prueba"+
          "\nNumero de enlaces del tercer parrafo = "+enlaces.length);

    }
</script>
```

## LABORATORIO 1

### Trabajando con Arrays

Se pide diseñar una página HTML donde ejecute las siguientes operaciones:

- Ejecutar el carrusel de imágenes visualizando el proceso en la imagen de la cabecera de la página
- Ejecutar un método donde muestre el calendario correspondiente al mes del sistema.

### 1. Creando el Sitio Web

Defina una carpeta llamado SitioWeb\_05, en ella crea las siguientes subcarpetas, tal como se muestra.

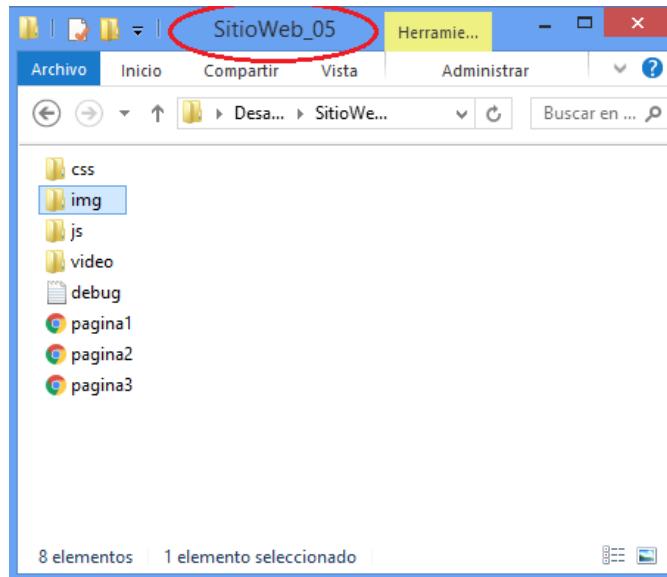


Figura 110 : Caso práctico  
Fuente.- Elaboración Propia

### 2. Trabajando con Visual Studio Code

Para crear una página web, abrir el aplicativo Visual Studio Code.

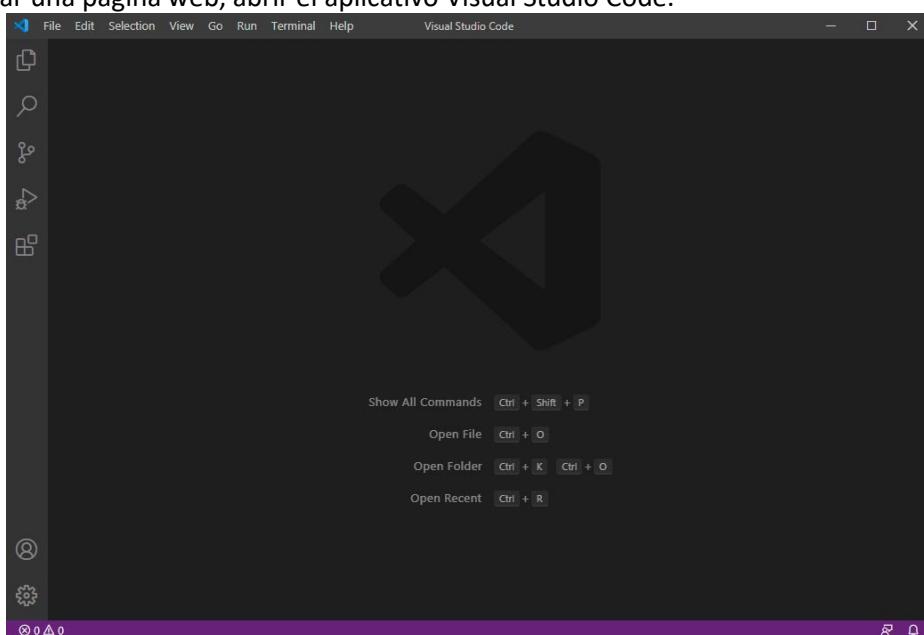


Figura 111: Caso práctico  
Fuente.- Elaboración Propia

Para trabajar con la carpeta llamada SitioWeb\_05 en Visual Studio Code, arrastre la carpeta hacia la ventana del Visual, tal como se muestra; y luego soltarla.

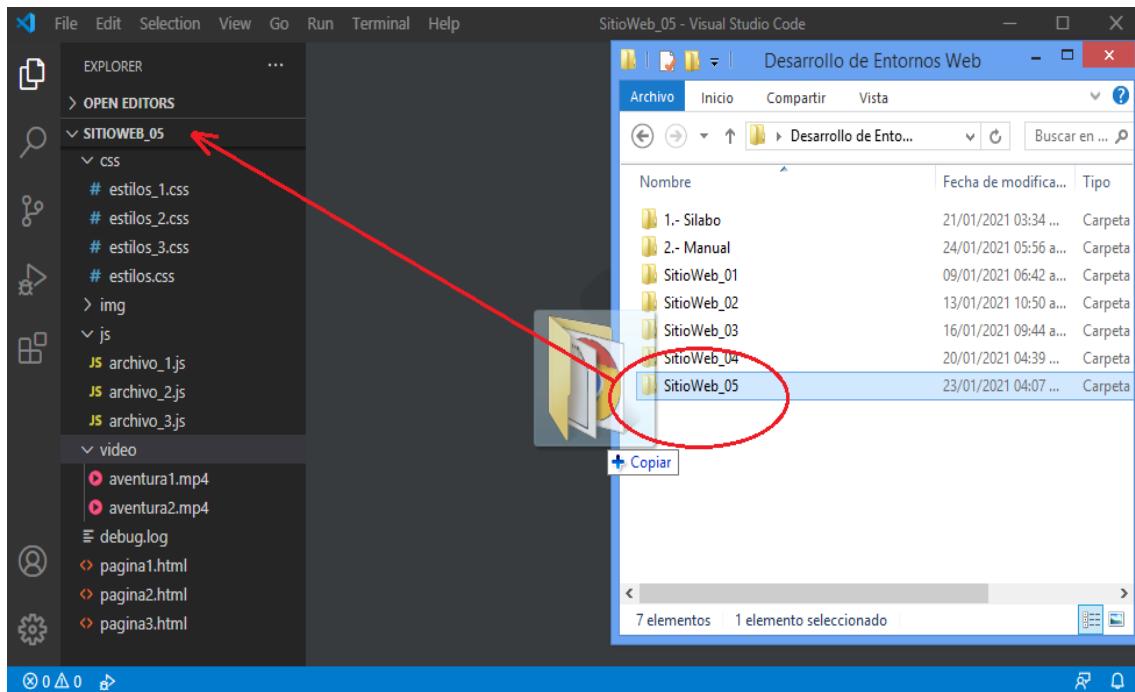


Figura 112 : Caso práctico  
Fuente.- Elaboración Propia

Donde se visualiza la carpeta en la parte izquierda de la ventana del aplicativo

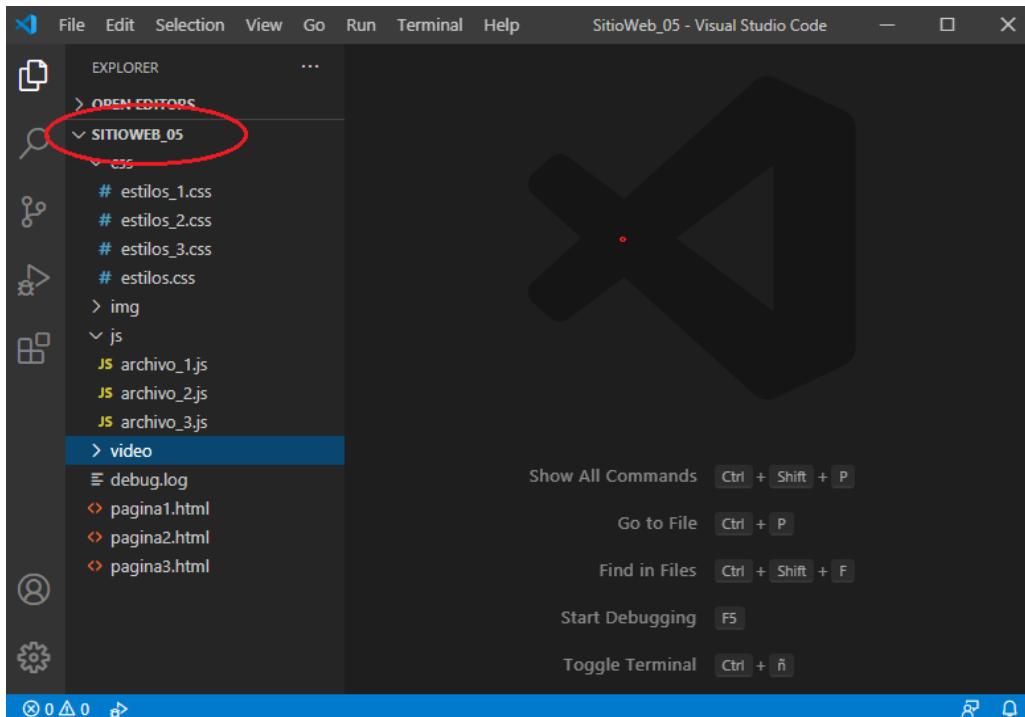


Figura 113: Caso práctico  
Fuente.- Elaboración Propia

## 1. Diseñando la página web

En esta etapa vamos a diseñar la página1.html, tal como se muestra.



Figura 114: Caso práctico  
Fuente.- Elaboración Propia

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_1.css y al archivo JS: archivo\_1.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

① pagina1.html X
② pagina1.html > ③ html > ④ body > ⑤ div#principal
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <title></title>
5       <meta charset="utf-8">
6       <link href="css/estilos_1.css" rel="stylesheet" />
7     </head>
8     <body>
9       <div id="principal">
10         <nav> ...
11         </nav>
12         <header> ...
13         </header>
14         <section> ...
15         </section>
16         <aside id="aside_cal">
17           </aside>
18         <footer> ...
19         </footer>
20       </div>
21     </body>
22   </html>
23   <script src="js/archivo_1.js"></script>
24 
```

Ln 9, Col 25 Tab Size: 4 UTF-8 with BOM CRLF HTML ⚙

Enlazando al archivo css a través de la etiqueta <link>

Enlazando al archivo js a través de la etiqueta <script>

Figura 115 : Caso práctico  
Fuente.- Elaboración Propia

### Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```

<body>
    <div id="principal">
        <nav>
            <a class="item" href="#">Home</a>
            <a class="item" href="pagina2.html">Productos</a>
            <a class="item" href="#">Lanzamientos</a>
            <a class="item" href="#">Testimonios</a>
            <a class="item" href="#">Servicios</a>
            <a class="item" href="#">Contactenos</a>
        </nav>
        <header>
            <h1 id="h1cab">Conociendo nuestras tradiciones en el Perú</h1>
            
        </header>
        <section>
            <h2 class="h2cab">Tradiciones y Costumbres</h2>
            <article class="blok">
                
            </article>
            <article class="blok">
                
            </article>
            <article class="blok">
                
            </article>
            <article class="blok">
                
            </article>
        </section>
        <aside id="aside_cal">

        </aside>
        <footer>
            
        </footer>
    </div>
</body>

```

### Definiendo el archivo estilos\_1.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```

body{
    background-color: gray;
}
#principal{
    width: 80%;
    float: left;
    margin: 0 10% 0 10%;
    border-radius: 20px;
}

```

```
        background-color: white;
    }
header,footer{
    width: 100%;
    float: left;
    margin-bottom: 5px;
}
section{
    width: 70%;
    float: left;
    margin-bottom: 5px;
}
#aside_cal{
    width: 30%;
    height: auto;
    float: left;
    margin-bottom: 5px;
}
table{
    margin: 0 auto;
}
td{
    text-align: center;
}
.td_dia{
    text-align: center;
    background-color: blue;
    color:white;
}
#pcab{
    text-align:justify;
    font-size: 18px;
    visibility: hidden;
    height: 0px;
}
nav{
    width: 90%;
    margin:0 5% 0 5%;
    height: auto;
    float: left;
    text-align:right;
}
.item{
    width:15%;
    margin:10px 0 10px 0;
    height:20px;
    float:left;
    color:gray;
    text-align:center;
    text-decoration:none;
}
#h1cab{
```

```
        text-align: center;
        color:gray;
    }
#imgcab{
    width: 100%;
    height: 250px;
    float: left;
    margin:0;
}
.h2cab {
    text-align: center;
    color: gray;
}
#ph2{
    text-align:justify;
    color:gray;
    margin-right:49%;
    margin-left:1%;
    float:left;
}
.blok{
    width: 22%;
    min-height: 200px;
    height: 255px;
    border:1px solid;
    float: left;
    margin:1%;
    color: gray;
    overflow: hidden;
}
.imgblok{
    width:98%;
    height:250px;
    margin:3px 1% 3px 1%;
    float:left;
}
#imgpie{
    width: 100%;
    height: 70px;
    float: left;
    margin-top:20px;
    margin-bottom: 20px;
}
```

### Definiendo la programación.

1. En el archivo de JavaScript archivo\_1.js, defina los Array imgs, el cual almacena la lista de las imágenes para la función carrusel(); y el Array meses, el cual almacena los meses del año expresados en letras, tal como se muestra.

The screenshot shows a code editor interface with three tabs at the top: 'pagina1.html', '# estilos\_1.css', and 'JS archivo\_1.js X'. The 'archivo\_1.js' tab is active. The code area contains the following lines of JavaScript:

```

js > JS archivo_1.js > ...
1 var imgs=new Array("img/img1.jpg","img/img2.jpg","img/img3.jpg","img/img4.jpg");
2
3 var meses=new Array("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","Agosto",
4 "Setiembre","Octubre","Noviembre","Diciembre");
5
6 var c=0;
7

```

At the bottom right of the code editor, there is status information: 'Ln 53, Col 1 Tab Size:4 UTF-8 CRLF JavaScript ⚡ ⚡'.

Figura 116: Caso práctico

Fuente.- Elaboración Propia

2. Defina la función carrusel(), donde cambia la imagen de la cabecera por cada segundo, uso del setTimeout

The screenshot shows the same code editor interface as Figure 116. The 'archivo\_1.js' tab is active. The code now includes the 'carrusel()' function definition:

```

js > JS archivo_1.js > ...
1 var imgs=new Array("img/img1.jpg","img/img2.jpg","img/img3.jpg","img/img4.jpg");
2
3 var meses=new Array("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","Agosto",
4 "Setiembre","Octubre","Noviembre","Diciembre");
5
6 var c=0;
7
8 function carrusel(){
9     c++;
10    if(c>=4) {c=0;}
11    document.getElementById("imgcab").setAttribute("src",imgs[c]);
12
13    setTimeout("carrusel()",1000);
14 }
15

```

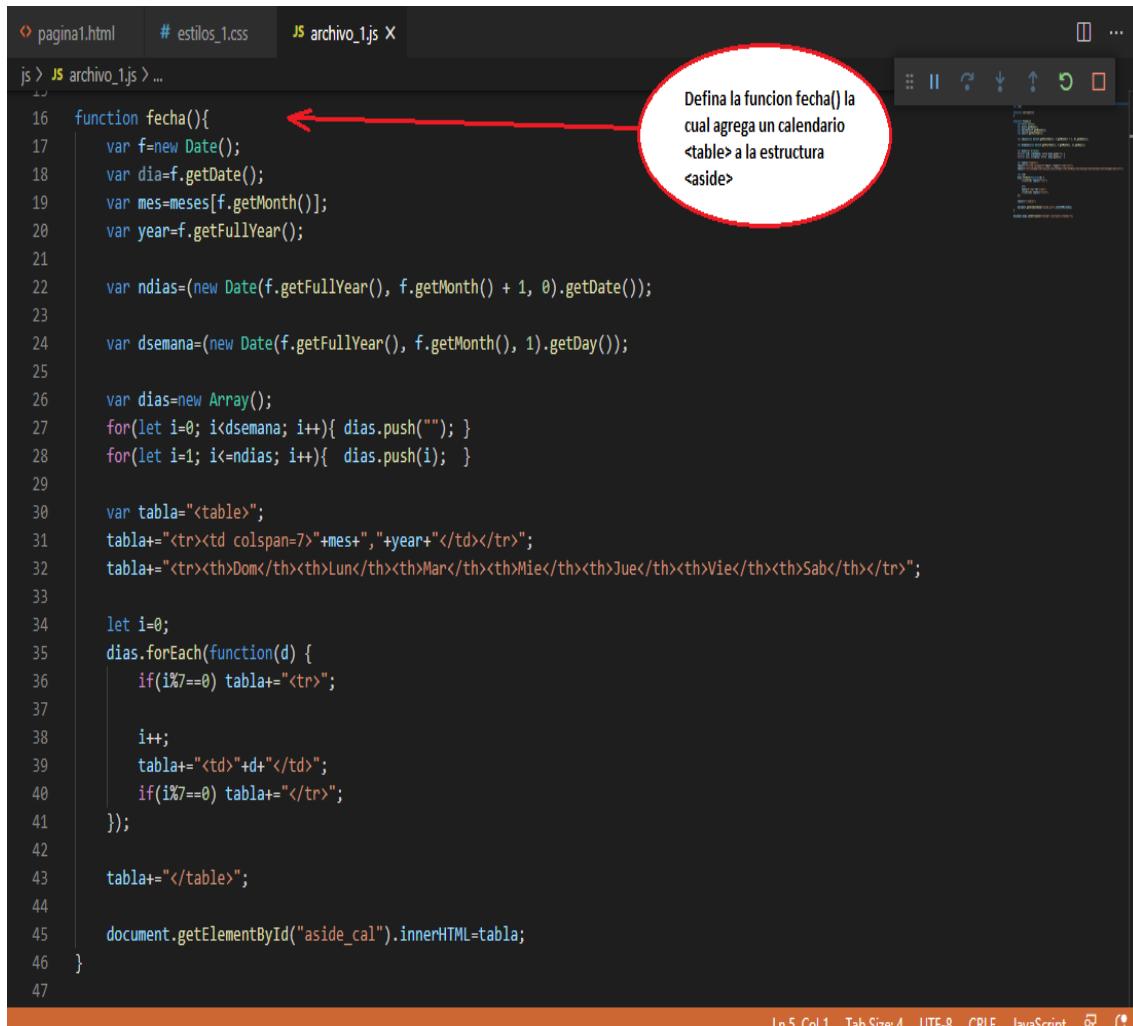
A red arrow points from the text 'Defina la función carrusel(), la cual cambia la imagen de la cabecera (imgcab) por cada segundo' to the opening brace of the 'carrusel()' function definition.

At the bottom right of the code editor, there is status information: 'Ln 5, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚡ ⚡'.

Figura 117: Caso práctico

Fuente.- Elaboración Propia

3. Defina la función fecha(), la cual imprime en la estructura <aside>, aside\_cal, una tabla, donde se visualiza el calendario del mes. En este proceso debemos definir el número de días del mes (ndias) y el dia de la semana del primer dia del mes (dsemana). A partir de estas variables defina el Array días(), la cual se agrega los días (expresados en números) del mes correspondiente, procediendo luego con un foreach agrega las celdas <td> a cada dia del mes.



```

16  function fecha(){
17      var f=new Date();
18      var dia=f.getDate();
19      var meses=meses[f.getMonth()];
20      var year=f.getFullYear();
21
22      var ndias=(new Date(f.getFullYear(), f.getMonth() + 1, 0).getDate());
23
24      var dsemana=(new Date(f.getFullYear(), f.getMonth(), 1).getDay());
25
26      var dias=new Array();
27      for(let i=0; i<dsemana; i++){ dias.push(""); }
28      for(let i=1; i<=ndias; i++){ dias.push(i); }
29
30      var tabla=<table>;
31      tabla+=<tr><td colspan=7>+mes+ , +year+</td></tr>;
32      tabla+=<tr><th>Dom</th><th>Lun</th><th>Mar</th><th>Mie</th><th>Jue</th><th>Vie</th><th>Sab</th></tr>;
33
34      let i=0;
35      dias.forEach(function(d) {
36          if(i%7==0) tabla+=<tr>;
37
38          i++;
39          tabla+=<td>+d+</td>;
40          if(i%7==0) tabla+=</tr>;
41      });
42
43      tabla+=</table>;
44
45      document.getElementById("aside_cal").innerHTML=tabla;
46  }

```

Figura 118 : Caso práctico  
Fuente.- Elaboración Propia

4. Para finalizar, enlazar al evento load, las funciones definidas, tal como se muestra

```

pagina1.html # estilos_1.css JS archivo_1.js

js > JS archivo_1.js > ...
1 var imgs=new Array("img/img1.jpg","img/img2.jpg","img/img3.jpg","img/img4.jpg");
2
3 var meses=new Array("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","Agosto",
4 "Setiembre","Octubre","Noviembre","Diciembre");
5
6 var c=0;
7
8 > function carrusel(){ ... }
9
10 > function fecha(){ ... }
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48 document.body.setAttribute("onload", "carrusel();fecha()");
49

```

Figura 119 : Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina1.html, y presiona la tecla F5.



Figura 120: Caso práctico  
Fuente.- Elaboración Propia

# Resumen

1. Un arreglo es como una variable donde podemos introducir varios valores, en lugar de solamente uno como ocurre con las variables normales.
2. La sentencia para crear un objeto array vacío es: var miArray = new Array().
3. Para crear el array especificando el número de compartimentos que va a tener es  
`var miArray = new Array(10);`
4. En las casillas de los arrays podemos guardar datos de cualquier tipo.
5. Para declarar un array y cargar valores en un mismo paso se escribe:

```
var arrayRapido = [12.3, 57,"Hilda","25/11/1990"].
```

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://www.desarrolloweb.com/articulos/631.php>
- [https://msdn.microsoft.com/es-es/library/k4h76zbx\(v=vs.94\).aspx](https://msdn.microsoft.com/es-es/library/k4h76zbx(v=vs.94).aspx)
- <https://medium.com/@rodrwan/arreglos-en-JavaScript-4c9164e12a16>
- [https://www.ecured.cu/Arreglos\\_en\\_JavaScript](https://www.ecured.cu/Arreglos_en_JavaScript)
- <https://desarrolloweb.com/articulos/630.php>
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos\\_globales/Array](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array)

## 4.2. CHILDNODES

Los navegadores web representan la página web en forma de árbol de nodos, por tanto, tendremos que saber cómo acceder a dichos nodos para crear efectos dinámicos (cambios en los nodos) con JavaScript.

Para ello JavaScript usa términos como childNodes, nodeType, nodeName, nodeValue, firstChild, lastChild, parentNode, nextSibling, previousSibling, etc. A continuación, mostramos la lista de estos términos:

Palabra clave	Significado	Ejemplo aprenderaprogramar.com
parentNode	Nodo padre de un nodo	...childNodes[1].childNodes[3].parentNode
childNodes	Array conteniendo los hijos de un nodo	document.childNodes[1].childNodes[1]
firstChild	Primer hijo de un nodo (empezando por la izquierda)	document.firstChild
lastChild	Ultimo hijo de un nodo (el más a la derecha)	document.childNodes[1].lastChild
nextSibling	Próximo nodo hermano (situado a la derecha)	document.childNodes[1].nextSibling
previousSibling	Anterior nodo hermano (situado a la izquierda)	...childNodes[2].childNodes[0].previousSibling

### 4.2.1. Trabajando con childNodes

La propiedad childNodes devuelve una colección de nodos hijos de un nodo, como un objeto NodeList.

Los nodos de la colección están ordenados como aparecen en el código fuente y se puede acceder por los números de índice. El índice comienza en 0.

Para determinar el número de nodos hijo, utilizamos longitud de la propiedad del objeto NodeList, entonces se puede recorrer todos los nodos secundarios y extraer la información que deseé.

**Nota:** El espacio en blanco en el interior de los elementos es considerado como texto, y el texto se considera como nodos. Los comentarios también son considerados como nodos. Para devolver una colección de nodos de elemento de un nodo (excluir text y comentarios), utilice la propiedad child.

#### 4.2.2. Métodos para acceder a nodos hijos

##### parentNode

Por medio de parentNode podemos seleccionar el elemento padre de otro elemento. Por ejemplo, si tenemos el siguiente código:

```
<div>
  <p id="introduccion">Párrafo introductorio.</p>
</div>
```

la siguiente línea de script:

```
document.getElementById('introduccion').parentNode;
```

selecciona el elemento padre del elemento identificado como introduccion, en este caso el div.

##### firstChild

Con firstChild lo que seleccionamos es el primer hijo de un elemento. Supongamos el siguiente fragmento de código:

```
<div id="contenido">
  <p>Un párrafo.</p>
  <p>Otro párrafo.</p>
</div>
```

##### lastChild

La propiedad lastChild funciona exactamente como firstChild, pero se refiere al último de los hijos de un elemento. Se aplican, por tanto, las mismas indicaciones anteriores.

##### nextSibling

Seleccionar es el siguiente hermano de un elemento.

##### previousSibling

Funciona igual que nextSibling, pero selecciona el hermano anterior de un elemento.

##### hasChildNodes

Método que retorna un valor booleano que indica si tiene o no nodos hijos.

#### 4.2.3. Métodos para agregar o quitar nodos

appendChild: por medio de appendChild podemos agregar, en un nodo, un nuevo hijo, de esta manera:

```
elemento_padre.appendChild(nuevo_nodo);
```

El nuevo nodo se incluye inmediatamente después de los hijos ya existentes —si hay alguno— y el nodo padre cuenta con una nueva rama.

Por ejemplo, el siguiente código:

```
var lista = document.createElement('ul');
var item = document.createElement('li');
lista.appendChild(item);
```

Crea un elemento ul y un elemento li, y convierte el segundo en hijo del primero.

**insertBefore:** nos permite elegir un nodo del documento e incluir otro antes que él. Su sintaxis es:

```
elemento_padre.insertBefore(nuevo_nodo,nodo_de_referencia);
```

Si tuviéramos un fragmento de un documento como éste:

```
<div id="padre">
  <p>Un párrafo.</p>
  <p>Otro párrafo.</p>
</div>
```

Y quisiéramos añadir un nuevo párrafo antes del segundo, lo haríamos así:

```
// Creamos el nuevo párrafo
var newp =
document.createElement('p').appendChild(document.createTextNode('Nuevo párrafo.'));

// Recojemos en una variable el segundo párrafo
var p2 = document.getElementById('padre').getElementsByName('p')[1];

// Y ahora lo insertamos
document.getElementById('padre').insertBefore(newp,p2);
```

**replaceChild:** Permite reemplazar un nodo por otro contamos con replaceChild, cuya sintaxis es:

```
elemento_padre.replaceChild(nuevo_nodo,nodo_a_reemplazar);
```

Con el mismo marcado que para el ejemplo de insertBefore, si quisiéramos sustituir el segundo párrafo por el que creamos, lo haríamos así:

```
document.getElementById('padre').replaceChild(nuevo_parrago,segundo_p);
```

**removeChild:** Si existe el método para agregar nodos, tiene sentido que podamos eliminarlos. Para ello existe el método removeChild. La sintaxis es:

```
elemento_padre.removeChild(nodo_a_eliminar);
```

Con el ejemplo anterior, eliminar el segundo párrafo, sería algo tan sencillo como:

```
document.getElementById('padre').removeChild(segundo_p);
```

**cloneNode:** podemos crear un clon de un nodo por medio de cloneNode:

```
elemento_a_clonar.cloneNode(booleano);
```

El booleano que se pasa como parámetro define si se quiere clonar el elemento —con el valor false—, o bien si se quiere clonar con su contenido —con el valor true—, es decir, el elemento y todos sus descendientes.

Si quisieramos clonar nuestro div de ejemplo con el siguiente código:

```
var clon = document.getElementById('padre').cloneNode(false);
```

clon contendría un elemento div, pero de esta manera:

```
var clon = document.getElementById('padre').cloneNode(true);
```

contendría un elemento div con dos párrafos que contendrían los mismos nodos de texto que el original.

## LABORATORIO 1

### Trabajando con ChildNodes

Se pide diseñar una página HTML donde ejecute las siguientes operaciones:

- Ejecutar un método donde muestre el calendario correspondiente al mes del sistema.  
Crear los elementos para este proceso

### 1. Creando el Sitio Web

Defina una carpeta llamado SitioWeb\_06, en ella crea las siguientes subcarpetas, tal como se muestra.

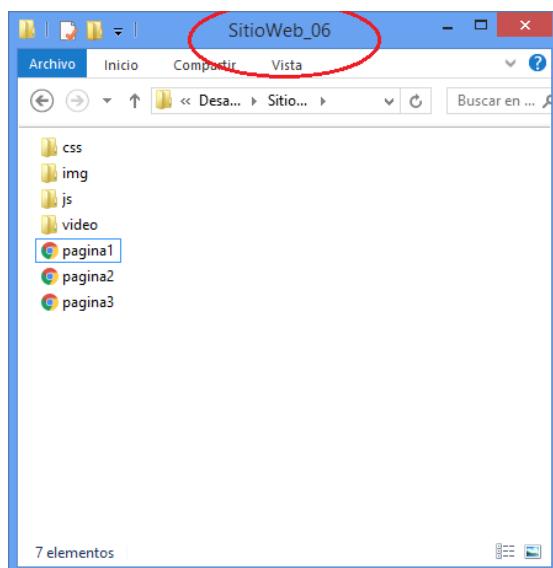


Figura 121: Caso práctico  
Fuente.- Elaboración Propia

### 2. Trabajando con Visual Studio Code

Para crear una página web, abrir el aplicativo Visual Studio Code.

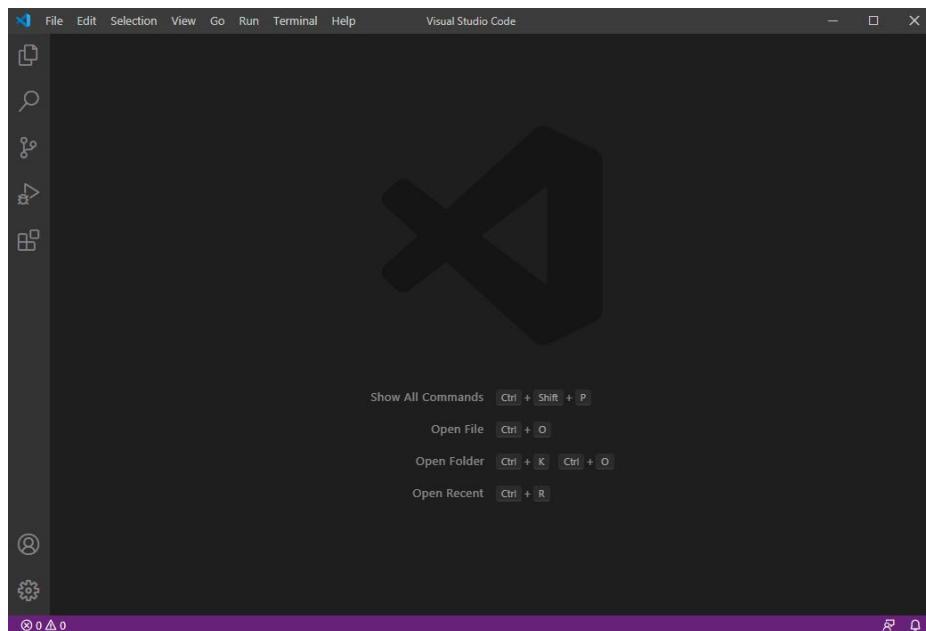


Figura 122 : Caso práctico  
Fuente.- Elaboración Propia

Para trabajar con la carpeta llamada SitioWeb\_06 en Visual Studio Code, arrastre la carpeta hacia la ventana del Visual, tal como se muestra; y luego soltarla.

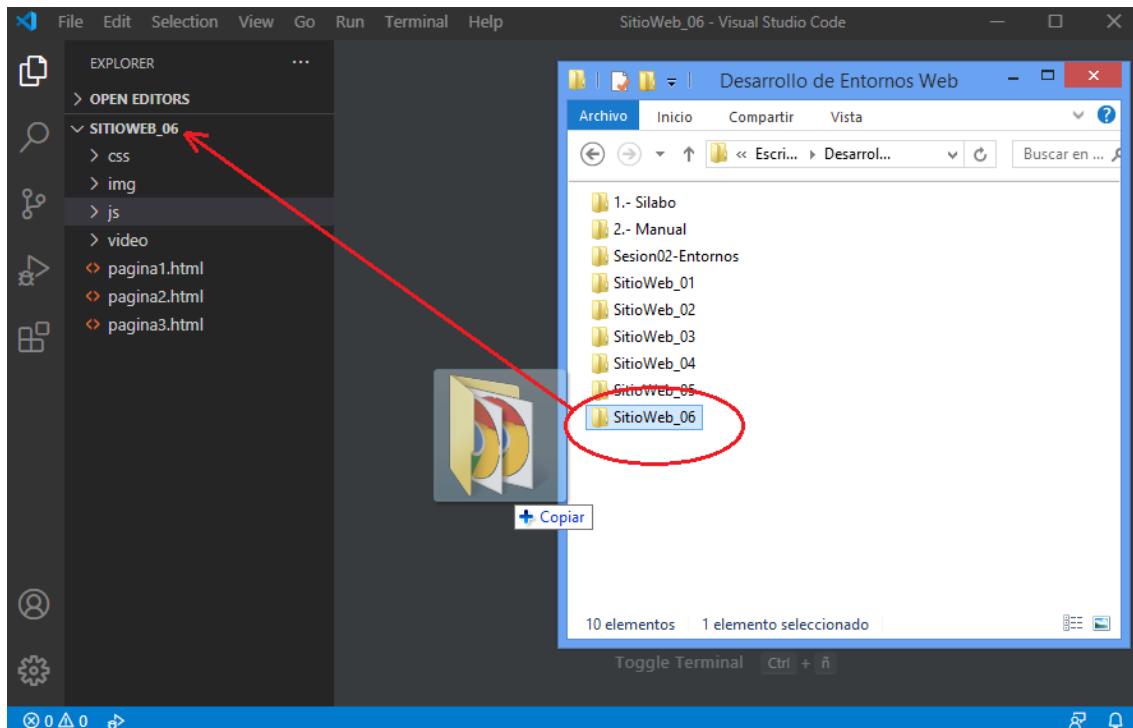


Figura 123 : Caso práctico  
Fuente.- Elaboración Propia

Donde se visualiza la carpeta en la parte izquierda de la ventana del aplicativo

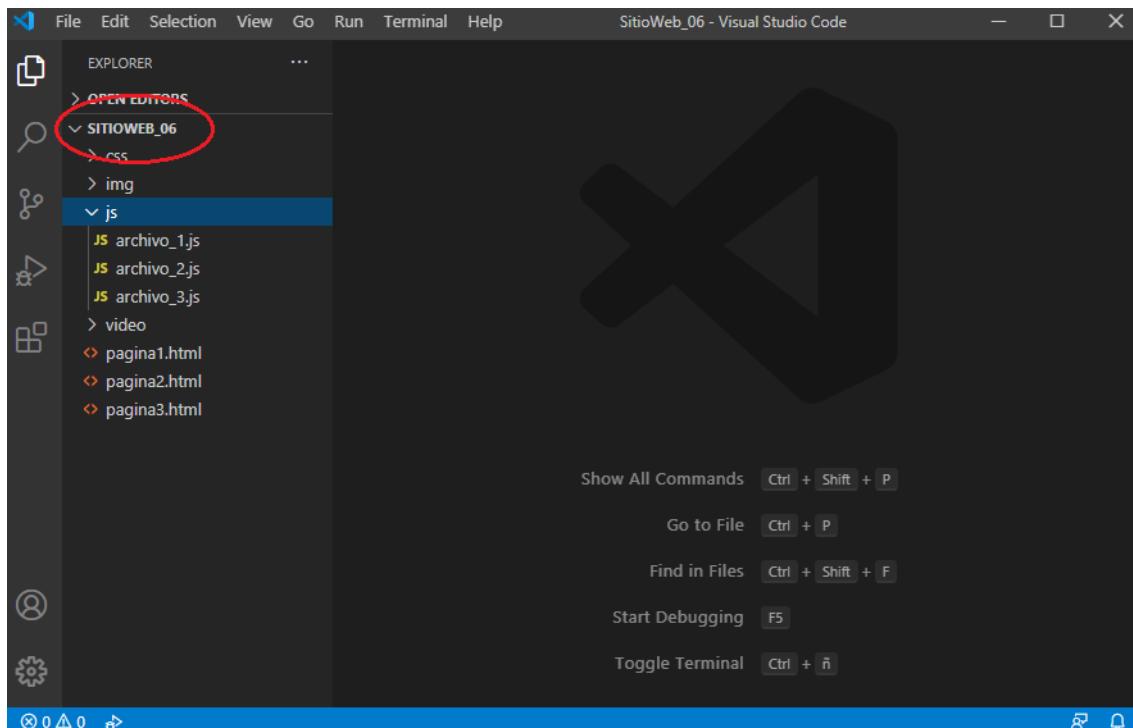


Figura 124 : Caso práctico  
Fuente.- Elaboración Propia

## 1. Diseñando la página web

En esta etapa vamos a diseñar la página1.html.

## Agregando los enlaces al archivo CSS y JavaScript

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_1.css y al archivo JS: archivo\_1.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title></title>
5   <meta charset="utf-8">
6   <link href="css/estilos_1.css" rel="stylesheet" />
7 </head>
8 <body>
9   <div id="principal">
10  <nav> ...
11  </nav>
12  <header> ...
13  </header>
14  <section> ...
15  </section>
16  <aside id="aside_cal">
17  </aside>
18  <footer> ...
19  </footer>
20 </div>
21 </body>
22 </html>
23 <script src="js/archivo_1.js"></script>
24

```

Figura 125: Caso práctico

Fuente.- Elaboración Propia

## Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```

<body>
  <div id="principal">
    <nav>
      <a class="item" href="#">Home</a>
      <a class="item" href="pagina2.html">Productos</a>
      <a class="item" href="#">Lanzamientos</a>
      <a class="item" href="#">Testimonios</a>
      <a class="item" href="#">Servicios</a>
      <a class="item" href="#">Contactenos</a>
    </nav>
    <header>
      <h1 id="h1cab">Conociendo nuestras tradiciones en el Perú</h1>
      
    </header>
    <section>
      <h2 class="h2cab">Tradiciones y Costumbres</h2>
      <article class="blok">
        
      </article>
      <article class="blok">
        
      </article>
    </section>
  </div>

```

```
</article>
<article class="blok">
    
</article>
<article class="blok">
    
</article>
</section>
<aside id="aside_cal">

</aside>
<footer>
    
</footer>
</div>
</body>
```

### Definiendo el archivo estilos\_1.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```
body{
    background-color: gray;
}
#principal{
    width: 80%;
    float: left;
    margin: 0 10% 0 10%;
    border-radius: 20px;
    background-color: white;
}
header, footer{
    width: 100%;
    height: auto;
    float: left;
    margin-bottom: 5px;
}
section{
    width: 70%;
    height: auto;
    float: left;
    margin-bottom: 5px;
}
#aside_cal{
    width: 30%;
    float: left;
    margin-bottom: 5px;
}
table{
    margin: 0 auto;
}
```

```
td{  
    text-align: center;  
}  
.td_dia{  
    text-align: center;  
    background-color: blue;  
    color:white;  
}  
#pcab{  
    text-align:justify;  
    font-size: 18px;  
    visibility: hidden;  
    height: 0px;  
}  
nav{  
    width: 90%;  
    margin:0 5% 0 5%;  
    float: left;  
    text-align:right;  
}  
.item{  
    width:15%;  
    margin:10px 0 10px 0;  
    height:20px;  
    float:left;  
    color:gray;  
    text-align:center;  
    text-decoration:none;  
}  
#h1cab{  
    text-align: center;  
    color:gray;  
}  
#imgcab{  
    width: 100%;  
    height: 250px;  
    float: left;  
    margin:0;  
}  
.h2cab {  
    text-align: center;  
    color: gray;  
}  
#ph2{  
    text-align:justify;  
    color:gray;  
    margin-right:49%;  
    margin-left:1%;  
    float:left;  
}  
.blok{
```

```

width: 22%;
min-height: 200px;
height: 255px;
border:1px solid;
float: left;
margin:1%;
color: gray;
overflow: hidden;
}
.imgblk{
width:98%;
height:250px;
margin:3px 1% 3px 1%;
float:left;
}
#imgpie{
width: 100%;
height: 70px;
float: left;
margin-top:20px;
margin-bottom: 20px;
}

```

### Definiendo la programación.

1. En el archivo de JavaScript archivo\_1.js, defina los Array dsemanas, el cual almacena los días de la semana; y el Array meses, el cual almacena los meses del año expresados en letras, tal como se muestra.

```

pagina1.html # estilos_1.css JS archivo_1.js X
js > JS archivo_1.js > fecha
1
2 var dsemanas=new Array("Dom","Lun","Mar","Mie","Jue","Vie","Sab");
3
4 var meses=new Array("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio",
5 "Agosto","Setiembre","Octubre","Noviembre","Diciembre");
6
7 > function fecha(){...
47 }

```

Arreglo de los días de la semana

Arreglo de los meses del año

función donde crea el calendario según mes y año

Figura 126 : Caso práctico  
Fuente.- Elaboración Propia

2. Defina la función fecha(), la cual define el dia, mes y año del sistema; luego almacena el número de días del mes (ndias) del mes y año y el dia de la semana del primer dia del mes (dsemana). A partir de estas variables defina el Array días(), la cual se agrega los días del mes correspondiente.

```

7  function fecha(){
8      var f=new Date();
9      var dia=f.getDate();
10     var mes=meses[f.getMonth()];
11     var year=f.getFullYear();
12
13     var ndias=(new Date(f.getFullYear(), f.getMonth() + 1, 0).getDate());
14
15     var dsemana=(new Date(f.getFullYear(), f.getMonth(), 1).getDay());
16
17     var dias=new Array();
18
19     for(let i=0; i<dsemana; i++){ dias.push(""); }
20     for(let i=1; i<ndias; i++){ dias.push(i); }

```

Figura 127: Caso práctico  
Fuente.- Elaboración Propia

3. A continuación, definimos un <table> y lo agregamos al bloque <aside>. A continuación, creamos un <tr> agregando al <table>, en esta sección agregamos cabeceras <th> a la fila creada. Y por último agregamos las filas y celdas para visualizar los días del mes y año correspondiente.

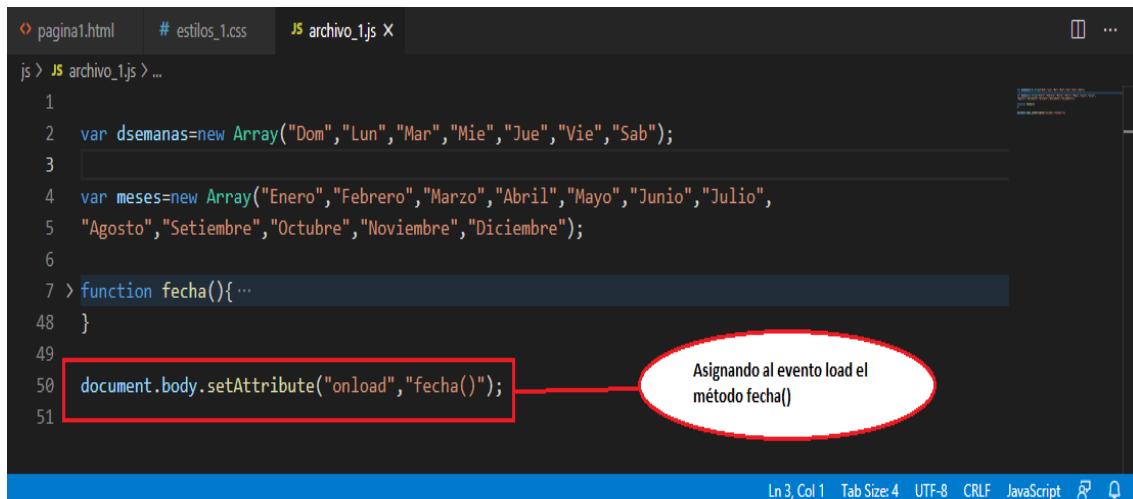
```

21
22 /*creando el table y agregado al aside */
23 var tabla=document.createElement("table");
24 document.getElementById("aside_cal").appendChild(tabla);
25
26 /*agregando la cabecera a la tabla */
27 var fila=document.createElement("tr");
28 tabla.appendChild(fila);
29
30 dsemanas.forEach(function(d){
31     let celda=document.createElement("th");
32     celda.innerHTML=d;
33     fila.appendChild(celda);
34 });
35
36 /*agregando los dias en la tabla */
37 let i=0;
38 dias.forEach(function(d) {
39     if(i%7==0){
40         fila=document.createElement("tr");
41         tabla.appendChild(fila);
42     }
43     i++;
44     let celda=document.createElement("td");
45     celda.innerHTML=d;
46     fila.appendChild(celda);
47 });
48 }

```

Figura 128 : Caso práctico  
Fuente.- Elaboración Propia

4. Para finalizar, enlazar al evento load, las funciones definidas, tal como se muestra



```

pagina1.html # estilos_1.css JS archivo_1.js X
js > JS archivo_1.js > ...
1
2 var dsemanas=new Array("Dom","Lun","Mar","Mie","Jue","Vie","Sab");
3
4 var meses=new Array("Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio",
5 "Agosto","Setiembre","Octubre","Noviembre","Diciembre");
6
7 > function fecha(){ ...
8 }
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50 document.body.setAttribute("onload", "fecha()");
51

```

Figura 129 : Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina1.html, y presiona la tecla F5.



Figura 130: Caso práctico  
Fuente.- Elaboración Propia

## LABORATORIO 2

### Trabajando con ChildNodes

Se pide diseñar una página HTML donde ejecute las siguientes operaciones:

- Ejecutar el carrusel de imágenes
- Ejecutar un método donde al seleccionar una tradición, visualice las imágenes de dicha tradición

#### 1. Diseñando la página web

En esta etapa vamos a diseñar la página2.html.

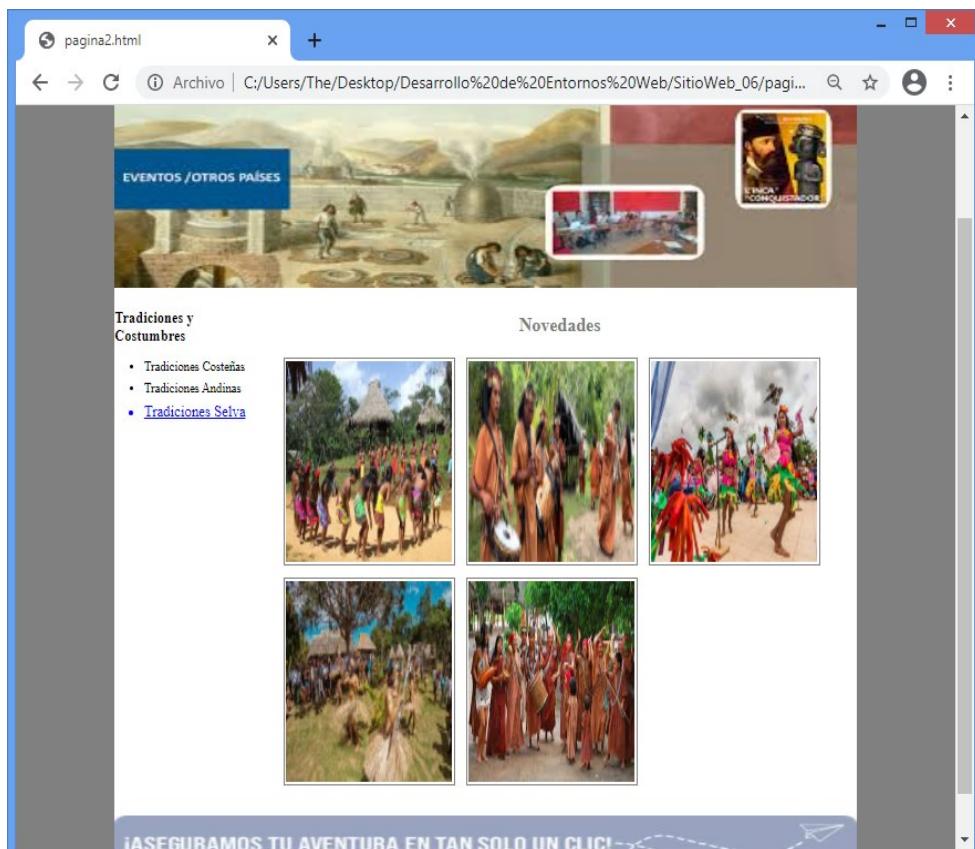


Figura 131 : Caso práctico

Fuente.- Elaboración Propia

### Agregando los enlaces al archivo CSS y JavaScript

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos\_2.css y al archivo JS: archivo\_2.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

![Screenshot of a code editor showing a portion of 'pagina2.html'. A red box highlights the line <link href=](js/archivo_2.js)

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5      <link href="css/estilos_2.css" rel="stylesheet" /> Enlazar a la hoja de estilo con la etiqueta <link>
6  </head>
7  <body>
8      <div id="principal">
9          <nav> ...
16         </nav>
17     <header> ...
21     </header>
22     <aside> ...
29     </aside>
30     <section> ...
35     </section>
36   <footer> ...
38   </footer>
39   </div>
40 </body>
41 </html>
42 <script src="js/archivo_2.js"></script> Enlazar el archivo JavaScript con la etiqueta <script>
43

```

Figura 132 : Caso práctico

Fuente.- Elaboración Propia

### Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```

<body>
    <div id="principal">
        <nav>
            <a class="item" href="pagina1">Home</a>
            <a class="item" href="#">Productos</a>
            <a class="item" href="#">Lanzamientos</a>
            <a class="item" href="#">Testimonios</a>
            <a class="item" href="#">Servicios</a>
            <a class="item" href="#">Contactenos</a>
        </nav>
        <header>
            <h1 id="h1cab">Productos Lego</h1>
            
        </header>
        <aside>
            <h3>Tradiciones y Costumbres</h3>
            <ul>
                <li class="li-tradicion" data-tradicion="costa">Tradiciones
Costeñas</li>
                <li class="li-tradicion" data-tradicion="andina">Tradiciones
Andinas</li>
                <li class="li-tradicion" data-tradicion="selva">Tradiciones
Selva</li>
            </ul>
        </aside>
        <section>
            <h2 class="h2cab">Novedades</h2>
            <main id="main-novedades">
            </main>
        </section>
    </div>

```

```
</section>
<footer>
    
</footer>
</div>
</body>
```

### Definiendo el archivo estilos\_2.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```
body{
    background-color: gray;
}
#principal{
    width: 80%;
    margin: 0 10% 0 10%;
    border-radius: 20px;
    background-color: white;
}
header, footer{
    width: 100%;
    margin-bottom: 5px;
}
aside{
    width: 20%;
    margin-bottom: 5px;
    float: left;
}
.li-tradicion{
    margin-top: 10px;
}
.li-tradicion:hover{
    text-decoration: underline;
    cursor: crosshair;
    color: blue;
    font-size: 1.2em;
}
section{
    width: 76%;
    padding: 5px 2% 5px 2%;
    margin-bottom: 5px;
    float: left;
}
nav{
    width: 90%;
    margin: 0 5% 0 5%;
    text-align: right;
    background-color: gray;
    margin-top: 1px;
}
```

```
.item{  
    width:15%;  
    margin:10px 0 10px 0;  
    display: inline-block;  
    height:20px;  
    color:yellow;  
    text-align:center;  
    text-decoration:none;  
}  
.item:hover{  
    color:blue;  
    text-decoration:underline;  
}  
#h1cab{  
    text-align: center;  
    color:gray;  
}  
#imgcab{  
    width: 100%;  
    height: 250px;  
    margin:0;  
}  
.h2cab {  
    text-align: center;  
    color: gray;  
}  
.blok{  
    width: 30%;  
    height: auto;  
    border:1px solid;  
    margin:1%;  
    color: gray;  
    float: left;  
    text-align: center;  
}  
.imgblok{  
    width:98%;  
    height:250px;  
    margin:3px 1% 3px 1%;  
    float:left;  
}  
#imgpie{  
    width: 100%;  
    height: 70px;  
    margin-top:20px;  
    margin-bottom: 20px;  
}  
#main-novedades{  
    width: 100%;  
    height: auto;  
    float: left;  
}
```

### Definiendo la programación.

- En el archivo de JavaScript archivo\_2.js, defina el Array imgs, el cual almacena las direcciones de las imágenes. Defina la función carrusel() donde cambia la imagen de la cabecera por cada segundo, tal como se muestra.

```

pagina2.html # estilos_2.css JS archivo_2.js X
js > JS archivo_2.js > ...
1 |
2 var imgs=new Array("img/img1.jpg","img/img2.jpg","img/img3.jpg","img/img4.jpg");
3
4 var c=0;
5 var ciclo;
6
7 function carrusel(){
8     document.getElementById("imgcab").setAttribute("src",imgs[c]);
9
10    c++;
11    if(c>4) {c=0;}
12    ciclo=setTimeout("carrusel()",1000);
13 }
14

```

definir el Array imgs, el cual almacena direcciones de imágenes

funcion carrusel(), donde cambia la imagen por cada segundo

Figura 133 : Caso práctico

Fuente.- Elaboración Propia

- Para el segundo proceso, defina el Array tradiciones, donde almacena los nombres de los archivos de imágenes. A continuación, defina la función novedades().

```

pagina2.html # estilos_2.css JS archivo_2.js ●
js > JS archivo_2.js > ...
1 var imgs=new Array("img/img1.jpg","img/img2.jpg","img/img3.jpg","img/img4.jpg");
2 var c=0;
3 var ciclo;
4
5 > function carrusel(){ ...
11 }
12
13 var tradiciones=new Array("costa1.jpg","costa2.jpg","costa3.jpg","costa4.jpg","costa5.jpg",
14 "andina1.jpg","andina2.jpg","andina3.jpg","andina4.jpg","andina5.jpg","andina6.jpg","andina7.jpg",
15 "selva1.jpg","selva2.jpg","selva3.jpg","selva4.jpg","selva5.jpg");
16
17 > function novedades(n){ ...
40 }
41

```

defina el Array tradiciones donde almacena los nombre de imágenes

defina la función novedades()

Figura 134: Caso práctico

Fuente.- Elaboración Propia

- Implementamos la función novedades(n), donde recupera el valor de data-tradicion, y a partir de allí, agregamos las imágenes correspondientes a la tradición seleccionada.

```

pagina2.html # estilos_2.css JS archivo_2.js
js > JS archivo_2.js > ...
16
17 function novedades(n){
18     let trad=document.getElementsByClassName("li-tradicion")[n].getAttribute("data-tradicion");
19
20     let arreglo=new Array();
21     tradiciones.forEach(function(t){
22         if(t.includes(trad)==true)
23             arreglo.push(t);
24     });
25
26     var seccion=document.getElementById("main-novedades");
27     while(seccion.hasChildNodes()){
28         seccion.removeChild(seccion.firstChild);
29     }
30     for(var i=0; i<arreglo.length; i++){
31         var blok=document.createElement("article");
32         blok.setAttribute("class","blok");
33         seccion.appendChild(blok);
34
35         var img=document.createElement("img");
36         img.setAttribute("class","imgblok");
37         img.setAttribute("src","img/tradiciones/"+arreglo[i]);
38         blok.appendChild(img);
39     }
40 }
41

```

La variable trad almacena el valor del atributo data-tradicion. En la variable arreglo, agregamos las imágenes que coincida con el valor de trad

Defina el objeto sección que almacena el elemento main-novedades, para quitar sus nodos hijos

A través de un proceso repetitivo, agregamos a la sección los elementos <article> y la imagen correspondiente a cada article

Ln 1, Col 81 Tab Size:4 UTF-8 CRLF JavaScript ⚙️ ⚙️

Figura 135: Caso práctico

Fuente.- Elaboración Propia

- Para finalizar, enlazar al evento load, las funciones definidas, tal como se muestra, así como al evento click a los elementos li-tradicion.

```

pagina2.html # estilos_2.css JS archivo_2.js
js > JS archivo_2.js > ...
12
13 var tradiciones=new Array("costa1.jpg","costa2.jpg","costa3.jpg","costa4.jpg","costa5.jpg",
14 "andina1.jpg","andina2.jpg","andina3.jpg","andina4.jpg","andina5.jpg","andina6.jpg","andina7.jpg",
15 "selva1.jpg","selva2.jpg","selva3.jpg","selva4.jpg","selva5.jpg");
16
17 > function novedades(n){ ...
40 }
41
42 document.body.setAttribute("onload","carrusel()");
43
44 for(let i=0; i<document.getElementsByClassName("li-tradicion").length;i++){
45     document.getElementsByClassName("li-tradicion")[i].addEventListener(
46         "click",function(){novedades(i)},false);
47 }

```

Enlazando el evento load con la función carrusel()

Enlazando el evento click a cada elemento li-tradicion con la función novedades

Ln 3, Col 11 Tab Size:4 UTF-8 CRLF JavaScript ⚙️ ⚙️

Figura 136 : Caso práctico

Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña pagina2.html, y presiona la tecla F5.

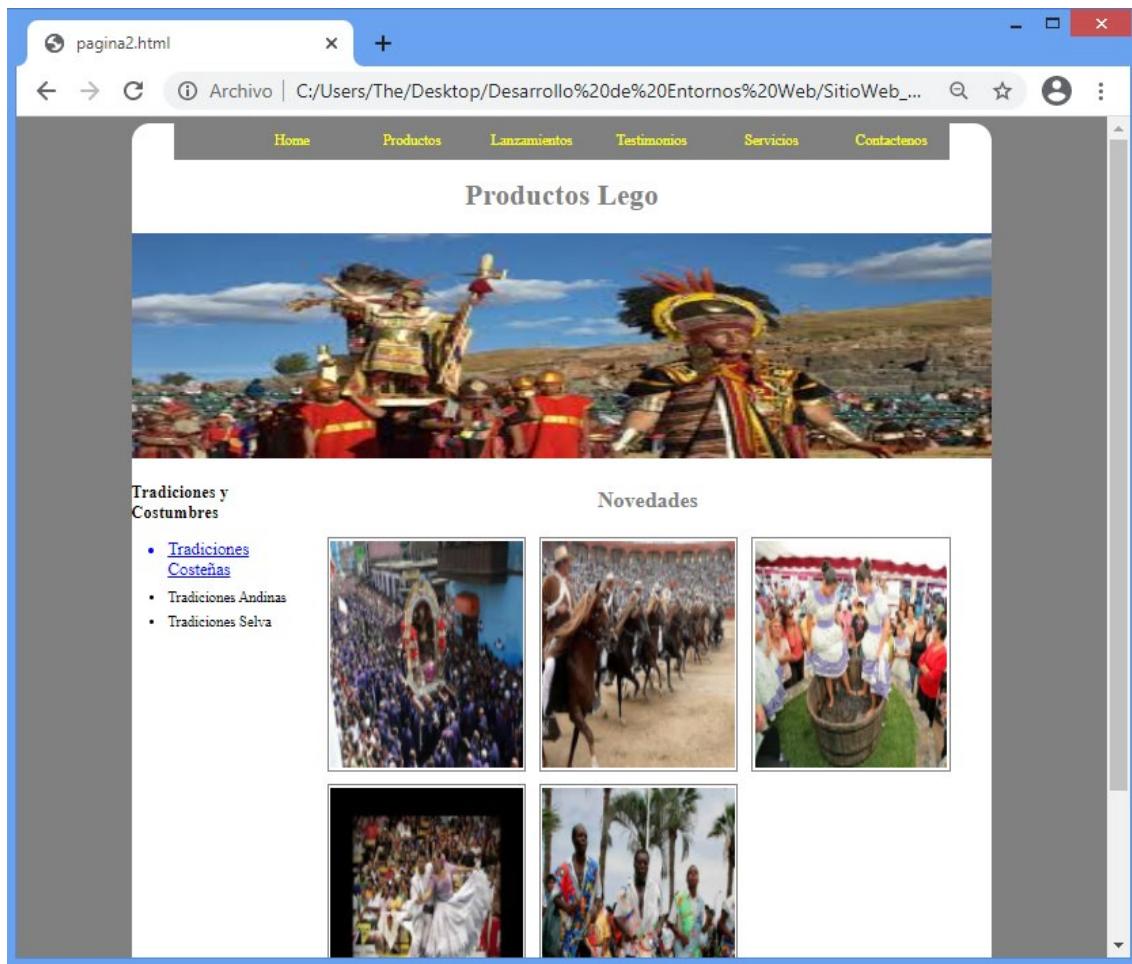


Figura 137: Caso práctico  
Fuente.- Elaboración Propia

# Resumen

1. La propiedad childNodes devuelve una colección de nodos hijos de un nodo, como un objeto NodeList.
2. Para recorrer los nodos secundarios, utilizamos longitud de la propiedad del objeto NodeList.
3. Los métodos para acceder a los nodos hijos: parentNode, firstChild, lastChild, nextSibling, previousSibling, hasChildNodes.
4. Los métodos para agregar o quitar nodos: appendChild, removeChild, insertBefore replaceChild, cloneChild.

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [http://www.codexempla.org/curso/curso\\_4\\_3\\_b.php](http://www.codexempla.org/curso/curso_4_3_b.php)
- [http://www.codexempla.org/curso/curso\\_4\\_3\\_d.php](http://www.codexempla.org/curso/curso_4_3_d.php)
- <https://developer.mozilla.org/en-US/docs/Web/API/Node/firstChild>
- [http://www.w3bai.com/es/jsref/prop\\_node\\_childnodes.html](http://www.w3bai.com/es/jsref/prop_node_childnodes.html)



# FORMULARIOS

---

## **LOGRO DE LA UNIDAD DE APRENDIZAJE**

Al término de la unidad, el alumno, implementa páginas HTML con formulario utilizando el lenguaje JavaScript, validando los datos con expresiones regulares.

## **TEMARIO**

### **5.1 Tema 7 : Formularios**

- 5.1.1 : El objeto form
- 5.1.2 : Objetos de un formulario
- 5.1.3 : Propiedades y eventos de un formulario
- 5.1.4 : Acceso a los elementos de un formulario

### **5.2 Tema 8 : Expresiones regulares**

- 5.2.1 : Introducción
- 5.2.2 : Definición de expresiones regulares
- 5.2.2.1 : Creación de expresiones
- 5.2.2.2 : Manejo de caracteres especiales

## **ACTIVIDADES PROPUESTAS**

- Los alumnos diseñan un formulario para logueo de usuario.
- Los alumnos diseñan un formulario para el ingreso de datos validándolos con expresiones regulares.

## 5.1. FORMULARIOS

La programación de aplicaciones, que contienen formularios web, siempre ha sido una de las tareas fundamentales de JavaScript. De hecho, una de las principales razones por las que se inventó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario. De esta forma, se evitaba recargar la página cuando el usuario cometía errores al llenar los formularios.

Los formularios en JavaScript nos permiten mucho más que recopilar datos para enviarlos a una dirección o programa de tratamiento. Puede servirnos también para interactuar con el usuario, ya que es la forma más fácil de recoger información, que nos servirá para que el usuario pueda variar algunos aspectos de la página.

### 5.1.1 el objeto form

Es un subobjeto del objeto document y este a su vez, lo es del objeto Window. Así como para crear una página en HTML se utilizan las etiquetas <HTML> Y </HTML>, lo mismo sucede con un formulario: el formulario debe estar contenido entre las etiquetas <form> y </form>

La sintaxis básica para referirnos a un formulario sería:

**document.forms.nombre\_formulario**

El objeto form posee las siguientes propiedades:

Propiedad	Descripción
name	Es el nombre único del formulario
action	Es el lugar al cual se envía el formulario para ser procesado. El action define la URL a la cual se envía dicho formulario.
method	Método de envío de los datos insertados en un formulario. El method puede ser:  GET = envía los datos en una cadena "visible". Conveniente para enviar pocos datos.  POST = envía los datos en forma "invisible". Conveniente para enviar una gran cantidad de datos.
target	Define la ventana o marco (frame) en la que se mostrarán o procesarán los resultados del formulario. El valor es el mismo que el utilizado en HTML (blank, self, top, nombre_marco, etc.)

Sintaxis:

```
<form name="nombre_formulario" action="procesar.asp" method="POST"
target="_blank">
.....campos....
</form>
```

El objeto form posee dos métodos:

Método	Descripción
submit	Envía el formulario.
reset	Restablece el formulario a los valores por defecto.

Sintaxis:

```
<form      name="nombre_formulario"      action="procesar.asp"      method="POST"
target="_blank">

.....campos....<br>

<input type="submit" value="enviar formulario">

<input type="reset" value="borrar">

</form>
```

### 5.1.2. Objetos de un formulario

Los objetos de un formulario (objetos del objeto form) son los "campos" de un formulario. En la siguiente tabla, podremos apreciar los tipos de objetos con su correspondiente descripción:

Objeto	Descripción
text	Campo de texto en el que los datos en él introducidos son visibles para el usuario.  <b>&lt;input type="text" name=".." value=".." size="..&gt;</b>
password	un campo de texto idéntico a text con la diferencia que los datos en él introducidos, no pueden ser visualizados por el usuario, sino que son mostrados con asteriscos *
	 <b>&lt;input type="password" name=".." value=".." size="..&gt;</b>
hidden	es un campo de texto oculto con un valor preestablecido y que el usuario no podrá visualizar en ningún momento.  <b>&lt;input type="hidden" name=".." value=".."&gt;</b>
textarea	es un campo de texto similar en su tratamiento a text, pero lo que en él varía es su apariencia, ya que puede tener un alto y ancho determinado, barras de scroll para navegar por su interior y admite saltos de línea.  <b>&lt;textarea rows=".." cols=".."&gt;...texto...&lt;/textarea&gt;</b>
radio	es un botón circular que permite elegir al usuario una opción de entre varias pertenecientes a un mismo grupo.  <b>&lt;input type="radio" name=".." value=".."&gt;</b>

checkbox	es una casilla de verificación que permite al usuario seleccionar más de una opción entre varias. Similar a radio, pero con esta ventaja.  <code>&lt;input type="checkbox" name=".." value=".."&gt;</code>
option	es una lista desplegable de varias opciones. Puede permitir la selección de una sola opción o de múltiples opciones (si se mantiene presionada la tecla Ctrl durante dicha selección).  <code>&lt;select name=".." multiple&gt; &lt;option value=".."&gt;texto1&lt;/option&gt; &lt;option value=".."&gt;texto2&lt;/option&gt; &lt;/select&gt;</code>
file	campo compuesto que permite examinar el disco duro para subir ficheros al servidor.  <code>input type="file" name=".." size="5"&gt;</code>
submit	es un tipo de botón que se encarga de enviar el formulario.  <code>&lt;input type="submit" value=".."&gt;</code>
reset	es un tipo de botón que se encarga de restablecer el formulario a sus valores por defecto.  <code>&lt;input type="reset" value=".."&gt;</code>
button	es un tipo de botón al que se le pueden asignar múltiples funciones mediante eventos.  <code>&lt;input type="button" value=".."&gt;</code>
image	es una imagen que actúa como botón (reset, button o submit)  <code>&lt;input type="image" src=".." ...&gt;</code>

### 5.1.3. Propiedades y eventos de los elementos de un formulario

Independientemente del método utilizado para obtener la referencia a un elemento de formulario, cada elemento dispone de las siguientes propiedades útiles para el desarrollo de las aplicaciones:

Propiedad	Descripción
type	Indica el tipo de elemento que se trata. Para los elementos de tipo <code>&lt;input&gt;</code> (text, button, checkbox, etc.) coincide con el valor de su atributo <code>type</code> . Para las listas desplegables normales (elemento <code>&lt;select&gt;</code> ) su valor es <code>select-one</code> , lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es <code>select-multiple</code> . Por último, en los elementos de tipo <code>&lt;textarea&gt;</code> , el valor de <code>type</code> es <code>textarea</code>
form	Es una referencia directa al formulario al que pertenece el elemento. Así, para acceder al formulario de un elemento, se puede utilizar <code>document.getElementById("id_del_elemento").form</code>
name	Obtiene el valor del atributo <code>name</code> de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar

value	Permite leer y modificar el valor del atributo value de XHTML. Para los campos de texto (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón
-------	---

Por último, los eventos más utilizados en el manejo de los formularios son los siguientes:

Evento	Descripción
onclick	Evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir XHTML (<input type="button">, <input type="submit">, <input type="image">).
onchange	Evento que se produce cuando el usuario cambia el valor de un elemento de texto (<input type="text"> o <textarea>). También se produce cuando el usuario selecciona una opción en una lista desplegable (<select>). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario pasa al siguiente campo del formulario, lo que técnicamente se conoce como que "el otro campo de formulario ha perdido el foco".
onfocus	Evento que se produce cuando el usuario selecciona un elemento del formulario.
onblur	Evento complementario de onfocus, ya que se produce cuando el usuario ha deseleccionado un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior "ha perdido el foco".

#### 5.1.4. Acceso a los elementos de un formulario

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios.

En primer lugar, cuando se carga una página web, el navegador crea automáticamente un array llamado **forms** y que contiene la referencia a todos los formularios de la página.

Para acceder al array **forms**, se utiliza el objeto **document**, por lo que **document.forms** es el array que contiene todos los formularios de la página. Como se trata de un array, el acceso a cada formulario se realiza con la misma sintaxis de los arrays. La siguiente instrucción accede al primer formulario de la página:

```
document.forms[0];
```

Además del array de formularios, el navegador crea automáticamente un array llamado **elements** por cada uno de los formularios de la página. Cada array **elements** contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario. Utilizando la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

```
document.forms[0].elements[0];
```

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

```
document.forms[0].elements[document.forms[0].elements.length-1];
```

Para acceder a un formulario, se realiza a través de su nombre (**atributo name**) o a través de su **atributo id**. El objeto document permite acceder directamente a cualquier formulario mediante su atributo name:

```
var formulario Principal = document. Formulario;
var formulario Secundario = document. otro_formulario;
```

```
<form name="formulario" >
...
</form>

<form name="otro formulario" >
...
</form>
```

Accediendo de esta forma a los formularios de la página, el script funciona correctamente, aunque se reordenen los formularios o se añadan nuevos formularios a la página. Los elementos de los formularios también se pueden acceder directamente mediante su atributo name:

```
var formularioPrincipal = document. formulario;
var primerElemento = document. formulario.elemento;

<form name="formulario">
<input type="text" name="elemento" />
</form>
```

Obviamente, también se puede acceder a los formularios y a sus elementos utilizando las funciones DOM de acceso directo a los nodos.

El siguiente ejemplo utiliza la habitual función **document.getElementById()** para acceder de forma directa a un formulario y a uno de sus elementos:

```
var formulario Principal = document.getElementById("formulario");
var primer Elemento = document.getElementById("elemento");

<form name="formulario" id="formulario" >
<input type="text" name="elemento" id="elemento" />
</form>.
```

## LABORATORIO 1

### Trabajando con Formularios

En este laboratorio vamos a crear una página HTML llamada inicio.html para ingresar el usuario y su clave, procediendo a visualizar con una alerta el nombre del usuario ingresado.

En este proceso realice las siguientes operaciones:

- a. Validar que los campos no estén vacíos
- b. Cuando el cursor se encuentre en uno del input, deberá de cambiar el color de fondo a amarillo

### 1. Creando el Sitio Web

Defina una carpeta llamado SitioWeb\_07, en ella crea las siguientes subcarpetas, tal como se muestra.

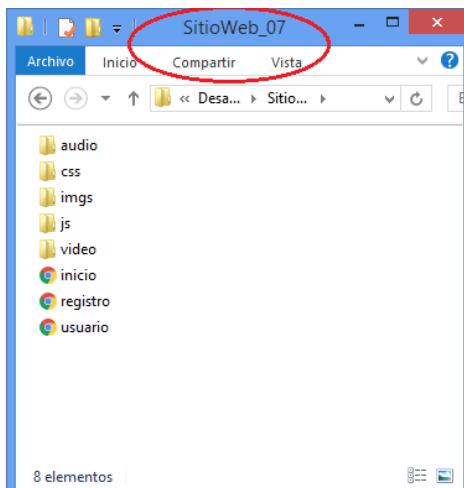


Figura 138 : Caso práctico  
Fuente.- Elaboración Propia

### 2. Trabajando con Visual Studio Code

Para crear una página web, abrir el aplicativo Visual Studio Code.

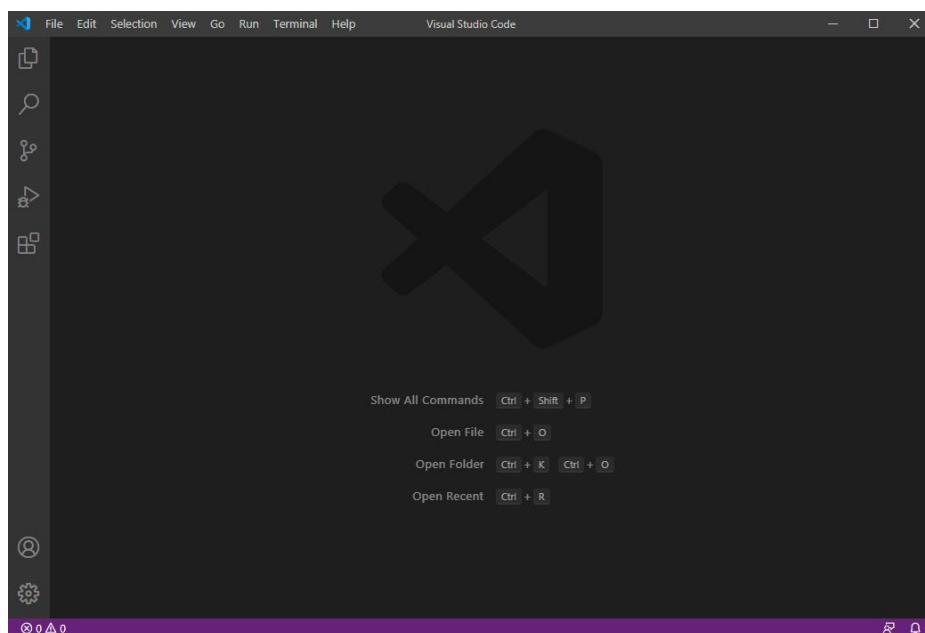


Figura 139 : Caso práctico  
Fuente.- Elaboración Propia

Para trabajar con la carpeta llamada SitioWeb\_07 en Visual Studio Code, arrastre la carpeta hacia la ventana del Visual, tal como se muestra; y luego soltarla.

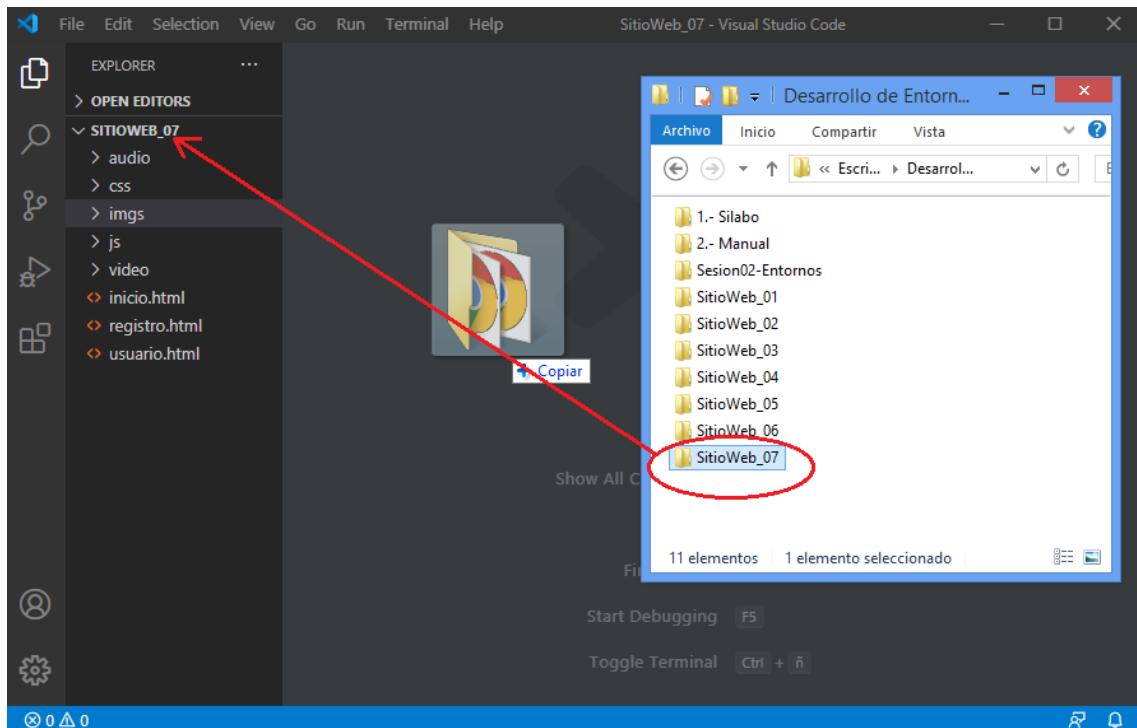


Figura 140 : Caso práctico  
Fuente.- Elaboración Propia

Donde se visualiza la carpeta en la parte izquierda de la ventana del aplicativo

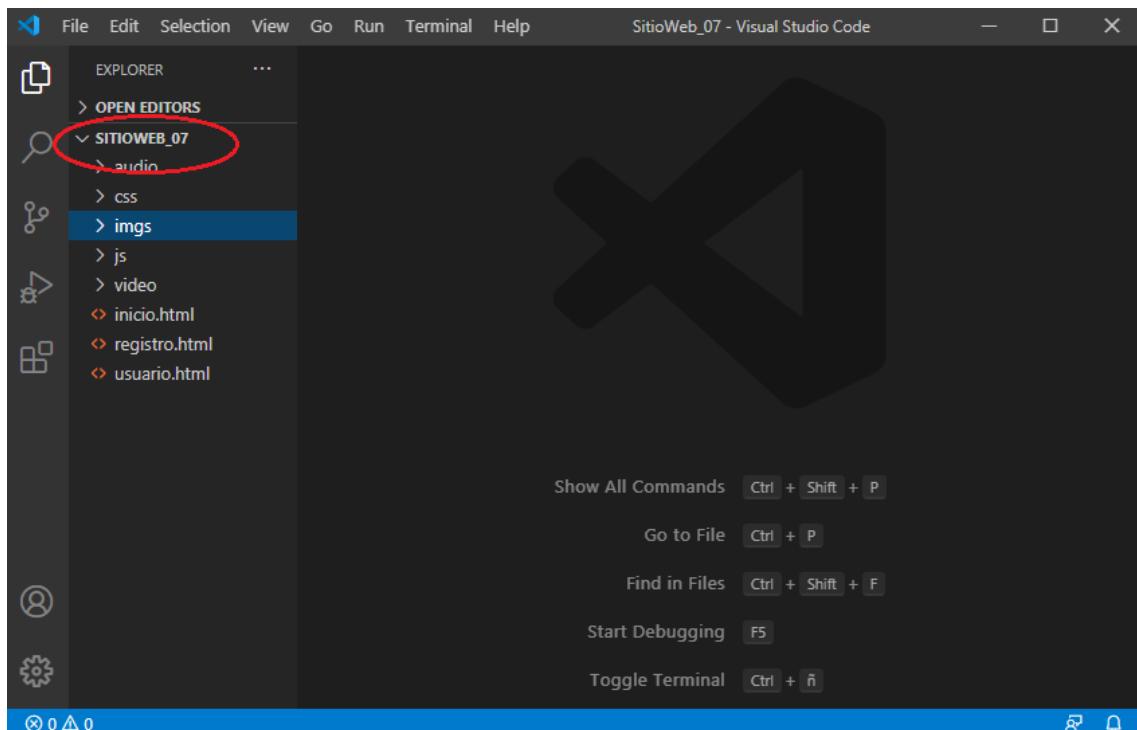


Figura 141 : Caso práctico  
Fuente.- Elaboración Propia

## 1. Diseñando la página web

En esta etapa vamos a diseñar la inicio.html.

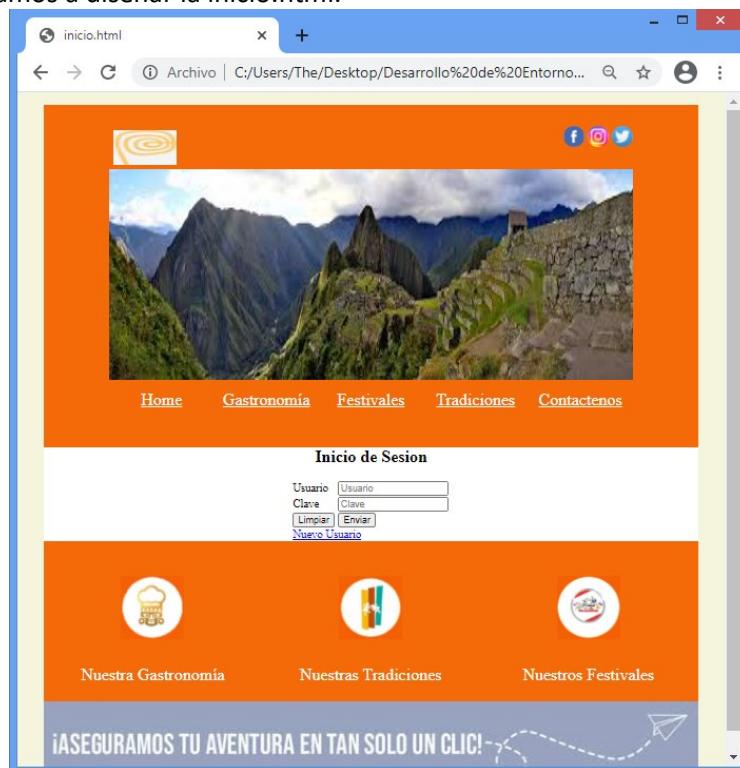


Figura 142 : Caso práctico

Fuente.- Elaboración Propia

### Agregando los enlaces al archivo CSS y JavaScript

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos.css y a los archivos JS: programa.js y validación.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

JS programa.js  inicio.html # estilos.css
  ...
  1  <!DOCTYPE html>
  2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
  3  <head>
  4    <meta charset="utf-8" />
  5    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  6    <title></title>
  7    <link rel="stylesheet" href="css/estilos.css" />          Enlazando al archivo CSS a
  8  </head>                                              través de la etiqueta <link>
  9  <body>
10    <main>
11      <header>...
12      </header>
13      <section id="slogueo">...
14      </section>
15      <footer>...
16      </footer>
17    </main>
18  </body>
19  </html>
20  <script src="js/programa.js"></script>          Enlazando a los archivos js, utilice
21  <script src="js/validacion.js"></script>           la etiqueta <script>
22

```

Figura 143: Caso práctico

Fuente.- Elaboración Propia

## Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```
<body>
  <main>
    <header>
      <div id="divlogo">
        
      </div>
      <div id="divsocial">
        
        
        
      </div>

      
      <div id="nav">
        <a href="#">Home</a>
        <a href="#">Gastronomía</a>
        <a href="#">Festivales</a>
        <a href="#">Tradiciones</a>
        <a href="#">Contactenos</a>
      </div>
    </header>
    <section id="slogueo">
      <h2>Início de Sesión</h2>
      <form id="frmlogueo" onsubmit="cargar(this)">
        <table>
          <tr>
            <td>Usuario</td>
            <td><input type="text" name="txtusuario" required placeholder="Usuario"> </td>
          </tr>
          <tr>
            <td>Clave</td>
            <td><input type="password" name="txtclave" required placeholder="Clave"></td>
          </tr>
          <tr>
            <td><input type="reset" value="Limpiar"></td>
            <td><input type="submit" value="Enviar"></td>
          </tr>
          <tr>
            <td colspan="2"><a href="usuario.html">Nuevo Usuario</a></td>
          </tr>
        </table>
      </form>
    </section>
    <footer>
      <div class="divfooter" >
        
        <p>Nuestra Gastronomía</p>
      </div>
      <div class="divfooter">
```

```

<p>Nuestras Tradiciones</p>
</div>
<div class="divfooter">
    
    <p>Nuestros Festivales</p>
</div>
    
</footer>
</main>
</body>
```

### Definiendo el archivo estilos.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```
body
{
    background-color:beige;
}

main{
    width:94%;
    margin-left:3%;
    margin-right:3%;
    margin-top: 10px;
    float:left;
    background-color:white;
    border-radius:20px;
}

header{
    width:80%;
    float:left;
    background-color: #F56A09;
    padding:30px 10% 30px 10%;
}

#imgheader{
    width:100%;
    height:300px;
    float:left;
}

#nav{
    width:100%;
    float:left;
    margin-top:5px;
}

#divlogo{
    width:40%;
```

```
float:left;
}

#divlogo>img{
  width:30%;
  height:50px;
  float:left;
  margin:2%
}
#divsocial{
  width:100%;
  text-align: right;
}

.social{
  width: 30px;
  width: 30px;
  border-radius: 20px;
}

#nav>a{
  height:40px;
  padding-top: 10px;
  padding-bottom: 10px;
  width: 20%;
  font-size:1.5em;
  color: white;
  text-align: center;
  float: left;
}

#nav>a:hover{
  background-color: white;
  color:#F56A09;
}

#slogueo>h2{
  text-align: center;
}

table{
  margin: 0 auto;
}

footer{
  width:100%;
  min-height: 200px;
  float:left;
  background-color: #F56A09;
}

.divfooter{
  width: 33.3%;
```

```

float: left;
text-align: center;
}

.divfooter>img{
width: 90px;
height: 90px;
padding-top: 50px;
padding-bottom: 10px;
}
.divfooter>p{
color:white;
font-size: 1.5em;
}

#imgfooter{
width: 100%;
height: 120px;
float: left;
}

```

### Definiendo la programación: programa.js.

1. En el archivo de JavaScript programa.js, defina el Array lista, el cual almacena las direcciones de las imágenes. Este Array() será utilizado en la función carrusel(), el cual cambia la imagen por intervalo de un segundo, tal como se muestra.

```

 1 var lista=new Array("imgs/header1.jpg","imgs/header2.jpg",
 2 | "imgs/header3.jpg", "imgs/header4.jpg");
 3
 4 var c=0;
 5
 6 function carrusel(){
 7     var img=document.getElementById("imgheader");
 8     c++;
 9     if(c>=lista.length) c=0;
10     img.src=lista[c];
11
12     setTimeout("carrusel()",1000);
13 }
14

```

Defina una Array llamado lista, la cual almacena las direcciones de las imágenes del carrusel

funcion carrusel(), cambia la imagen por cada segundo

Figura 144: Caso práctico  
Fuente.- Elaboración Propia

2. Defina las funciones cambio(e) y regresa(e), funciones que permiten cambiar la imagen de la red social. A continuación, defina las funciones gira(e) y retorna(e) funciones que permiten girar un elemento 360 grados y 0 grados en su eje Y por un segundo.

```

14
15  function cambio(e){
16      var img=e.getAttribute("data-social");
17      e.src="imgs/"+img+"1"+".jpg";
18  }
19
20  function regresa(e){
21      var img=e.getAttribute("data-social");
22      e.src="imgs/"+img+".jpg";
23  }
24
25  function gira(e){
26      e.style.transform="rotateY(360deg)";
27      e.style.transition="all 1s";
28  }
29
30  function retorna(e){
31      e.style.transform="rotateY(0deg)";
32      e.style.transition="all 1s";
33  }
34

```

función que permite cambiar la imagen de la red social utilizando el atributo data-social

función que permite visualizar la imagen original de la red social, utilizando el atributo data-social

función que permite girar un elemento 360 grados en el eje Y por 1 segundo

función que permite girar un elemento a 0 grados en el eje Y por un segundo

Figura 145 : Caso práctico

Fuente.- Elaboración Propia

3. A continuación, en el archivo programa.js, enlazamos a los elementos de las páginas, los eventos correspondientes, tal como se muestra.

```

29
30 > function retorna(e){ ...
31 }
32
33 /*eventos*/
34 document.body.setAttribute("onload", "carrusel()");
35
36 for(var i=0; i<document.getElementsByClassName("social").length; i++){
37 {
38     document.getElementsByClassName("social")[i].setAttribute("onmouseover", "cambio(this)");
39     document.getElementsByClassName("social")[i].setAttribute("onmouseout", "regresa(this)");
40 }
41
42 for(var i=0; i<document.getElementsByClassName("imgfooter").length; i++){
43     document.getElementsByClassName("imgfooter")[i].setAttribute("onmouseover", "gira(this)");
44     document.getElementsByClassName("imgfooter")[i].setAttribute("onmouseout", "retorna(this)");
45 }
46
47 /*fin de eventos*/
48
49

```

Enlazando al evento load, el método carrusel()

Enlazando a los elementos "social", los eventos mouseover y mouseout

Enlazando a los elementos imgfooter los eventos mouseover y mouseout

Figura 146 : Caso práctico

Fuente.- Elaboración Propia

### Definiendo la programación: validacion.js.

1. En el archivo de JavaScript validacion.js, defina dos variables que almacena los inputs de la página y enlazar los eventos focus y blur a cada uno de los elementos, tal como se muestra.

```

1 var usuario = document.querySelector('input[type="text"]');
2 var clave = document.querySelector('input[type="password"]');
3
4 usuario.addEventListener('focus', function(event){
5     event.target.style.background = 'blue';
6     event.target.style.color='yellow'
7 });
8
9 usuario.addEventListener('blur', function(event){
10    event.target.style.background = 'white';
11    event.target.style.color='black';
12 });
13
14 clave.addEventListener('focus', function(event){
15     event.target.style.background = 'blue';
16     event.target.style.color='yellow'
17 });
18
19 clave.addEventListener('blur', function(event){
20     event.target.style.background = 'white';
21     event.target.style.color='black';
22 });
23

```

definir las variables que almacena los input text y password

En usuario, defina el metodo del evento focus, donde al ingresar, cambia el color de fondo y de letra; y el metodo de evento blur, donde al salir se le asigna propiedades originales

En clave, defina el metodo de evento focus, donde al ingresar cambia el color de fondo y de letra; y el metodo de evento blur, donde al salir se le asigna sus propiedades originales

Figura 147: Caso práctico

Fuente.- Elaboración Propia

2. Defina dos Arrays() donde almacena los usuarios y sus claves, en la función cargar(frm) verificamos que los datos ingresados en los inputs, corresponde al usuario y su clave

```

1 var usuarios=["juan","carlos","ana","luisa","luis"];
2 var claves=["juan123","carlos123","ana123","luisa123","luis123"];
3
4 function cargar(frm){
5     var usu=frm.txtusuario.value;
6     let i=-1;
7     for(let j=0; j<usuarios.length; j++){
8         if(usuarios[j]==usu){
9             i=j; break;
10        }
11    }
12
13    if(i==-1){
14        window.event.preventDefault();
15        alert("Usuario no existe"); return;
16    }
17
18    var clave=frm.txtclave.value;
19    if(clave==claves[i]){
20        alert("Bienvenido,"+usu);
21        window.open("registro.html");
22    }
23    else{
24        window.event.preventDefault();
25        alert("Clave no coincide");
26    }
27 }

```

definir los Array() para almacenar los usuarios y sus claves

Defina la función cargar(frm) donde evalúa si el usuario y la clave se encuentran en los Arrays()

Figura 148 : Caso práctico

Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña inicio.html, y presiona la tecla F5.

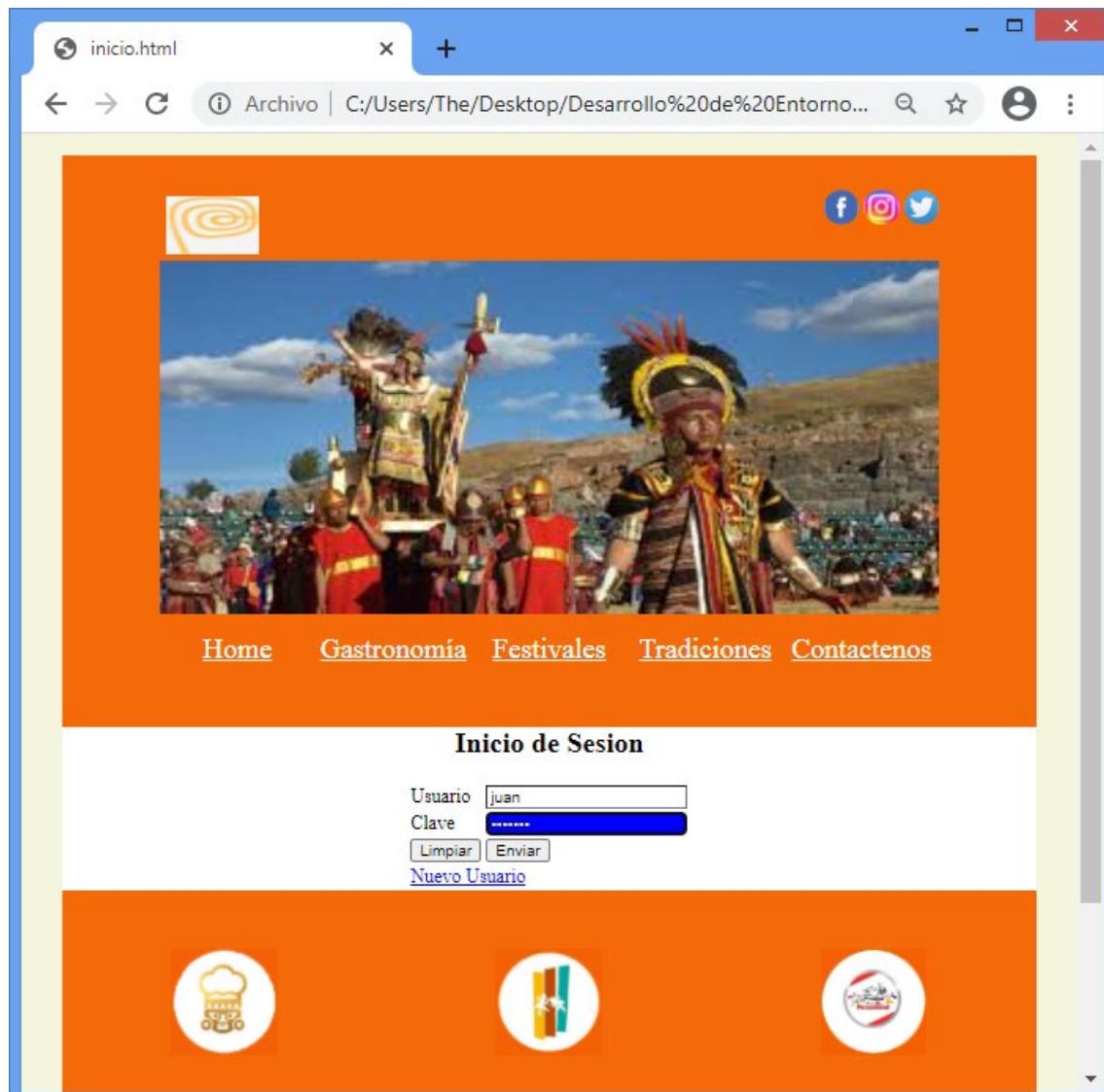


Figura 149 : Caso práctico  
Fuente.- Elaboración Propia

## LABORATORIO 2

### Trabajando con Formularios

En este laboratorio vamos a crear una página HTML llamada usuario.html para ingresar los datos del usuario, procediendo a verificar el ingreso de datos.

En este proceso realice las siguientes operaciones:

- Validar que los campos DNI, nombre y apellido no estén vacíos.
- Seleccionar una fecha, un grado de instrucción, género y por lo menos 1 curso

#### 1. Diseñando la página web

En esta etapa vamos a diseñar la usuario.html.

The screenshot shows a web browser window with the URL 'usuario.html?nombre=dedede&'. The page has a header with a logo, social media icons (Facebook, Instagram, Twitter), and navigation links: Home, Productos, Ofertas, Sedes, Contactenos. Below the header is a large orange banner featuring a photograph of people in traditional Andean ceremonial attire. The main content area is titled 'Registro de Usuario' and contains the following form fields:

Ingrese el DNI	<input type="text"/>
Ingrese el Nombre	<input type="text"/>
Ingrese su Apellido	<input type="text"/>
Fecha Nacimiento	<input type="text"/> dd/mm/aaaa
Grado de Instrucción	<input type="button" value="Superior"/>
Genero	<input checked="" type="radio"/> Masculino <input type="radio"/> Femenino
Cursos que domina	<input type="checkbox"/> Java <input type="checkbox"/> SQL Server <input type="checkbox"/> JavaScript <input type="checkbox"/> HTML <input type="checkbox"/> .NET
Comentarios	<input type="text"/>

At the bottom of the form are three buttons: Limpiar, Procesar, and Retornar.

Figura 150: Caso práctico  
Fuente.- Elaboración Propia

### Agregando los enlaces al archivo CSS y JavaScript

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos.css y a los archivos JS: programa.js y nuevo.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title></title>
7      <link rel="stylesheet" href="css/estilos.css" />
8  </head>
9  <body>
10     <main>
11         <header>...
12         </header>
13         <section id="soferta">...
14         </section>
15         <footer>...
16     </main>
17     </body>
18 </html>
19 <script src="js/programa.js"></script>
20 <script src="js/nuevo.js"></script>

```

Figura 151: Caso práctico  
Fuente.- Elaboración Propia

### Diseño de la página web

Defina los elementos de la página, tal como se muestra.

```

<body>
<main>
    <header>
        <div id="divlogo">
            
        </div>
        <div id="divsocial">
            
            
            
        </div>

        
        <div id="nav">
            <a href="#">Home</a>
            <a href="#">Gastronomía</a>
            <a href="#">Festivales</a>
            <a href="#">Tradiciones</a>
            <a href="#">Contactenos</a>
        </div>
    </header>
    <section id="slogueo">
        <h2 id="h2cab">Registro de Usuario</h2>
    </section>

```

```
<form id="frmusuario" onsubmit="nuevo(this)">
<table>
<tr>
    <td>Ingrese el DNI</td>
    <td><input name="dni" required placeholder="DNI"> </td>
</tr>
<tr>
    <td>Ingrese el Nombre</td>
    <td><input name="nombre" required placeholder="Nombre"> </td>
</tr>
<tr>
    <td>Ingrese su Apellido</td>
    <td><input name="apellido" required placeholder="Apellido"></td>
</tr>
<tr>
    <td>Fecha Nacimiento</td>
    <td><input type="date" name="fecha"></td>
</tr>
<tr>
    <td>Grado de Instrucción</td>
    <td><select name="grado">
        <option value="PR">Primaria</option>
        <option value="SE">Secundaria</option>
        <option value="SU" selected>Superior</option>
        <option value="MA">Maestría</option>
        <option value="DO">Doctorado</option>
    </select></td>
</tr>
<tr>
    <td>Genero</td>
    <td><input type="radio" name="sexo" value="M" checked>Masculino
        <input type="radio" name="sexo" value="F">Femenino
    </td>
</tr>
<tr>
    <td>Cursos que domina</td>
    <td>
        <input type="checkbox" name="curso" value="JA">Java
        <input type="checkbox" name="curso" value="SQL">SQL Server
        <input type="checkbox" name="curso" value="JS">JavaScript
        <input type="checkbox" name="curso" value="HTML">HTML
        <input type="checkbox" name="curso" value=".NET">.NET
    </td>
</tr>
<tr>
    <td>Comentarios</td>
    <td><textarea name="comentarios" cols="20" rows="10"></textarea></td>
</tr>
<tr>
    <td colspan="2"><input type="reset" value="Limpiar"><input type="submit" value="Procesar"><input type="button" value="Retornar" onclick="window.open('inicio.html')"> </td>
</tr>
```

```
</tr>
</table>
</form>
</section>
<footer>
  <div class="divfooter" >
    
    <p>Nuestra Gastronomía</p>
  </div>
  <div class="divfooter">
    
    <p>Nuestras Tradiciones</p>
  </div>
  <div class="divfooter">
    
    <p>Nuestros Festivales</p>
  </div>
  
</footer>
</main>
</body>
```

### Definiendo el archivo estilos.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```
body
{
  background-color:beige;
}

main{
  width:94%;
  margin-left:3%;
  margin-right:3%;
  margin-top: 10px;
  float:left;
  background-color:white;
  border-radius:20px;
}

header{
  width:80%;
  float:left;
  background-color: #F56A09;
  padding:30px 10% 30px 10%;
}

#imgheader{
  width:100%;
  height:300px;
```

```
float:left;
}

#nav{
    width:100%;
    float:left;
    margin-top:5px;
}
#divlogo{
    width:40%;
    float:left;
}
#divlogo>img{
    width:30%;
    height:50px;
    float:left;
    margin:2%
}
#divsocial{
    width:100%;
    text-align: right;
}
.social{
    width: 30px;
    width: 30px;
    border-radius: 20px;
}
#nav>a{
    height:40px;
    padding-top: 10px;
    padding-bottom: 10px;
    width: 20%;
    font-size:1.5em;
    color: white;
    text-align: center;
    float: left;
}
#nav>a:hover{
    background-color: white;
    color:#F56A09;
}
#slogueo>h2{
    text-align: center;
}
table{
    margin: 0 auto;
}

footer{
    width:100%;
    min-height: 200px;
    float:left;
```

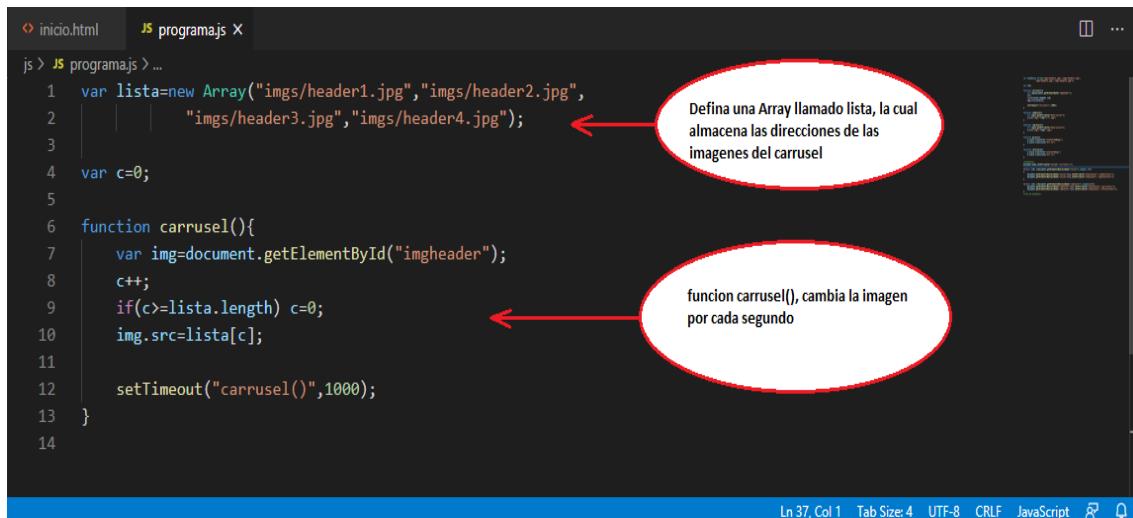
```

background-color: #F56A09;
}
.divfooter{
  width: 33.3%;
  float: left;
  text-align: center;
}
.divfooter>img{
  width: 90px;
  height: 90px;
  padding-top: 50px;
  padding-bottom: 10px;
}
.divfooter>p{
  color:white;
  font-size: 1.5em;
}
#imgfooter{
  width: 100%;
  height: 120px;
  float: left;
}

```

### Definiendo la programación: programa.js.

1. En el archivo de JavaScript programa.js, defina el Array lista, el cual almacena las direcciones de las imágenes. Este Array() será utilizado en la función carrusel(), el cual cambia la imagen por intervalo de un segundo, tal como se muestra.



```

// inicio.html      JS programa.js X
js > JS programa.js > ...
1 var lista=new Array("imgs/header1.jpg","imgs/header2.jpg",
2 | | | "imgs/header3.jpg","imgs/header4.jpg");
3
4 var c=0;
5
6 function carrusel(){
7   var img=document.getElementById("imgheader");
8   c++;
9   if(c>=lista.length) c=0;
10  img.src=lista[c];
11
12  setTimeout("carrusel()",1000);
13 }
14

```

Defina una Array llamado lista, la cual almacena las direcciones de las imágenes del carrusel

funcion carrusel(), cambia la imagen por cada segundo

Figura 152: Caso práctico  
Fuente.- Elaboración Propia

2. Defina las funciones cambio(e) y regresa(e), funciones que permiten cambiar la imagen de la red social. A continuación, defina las funciones gira(e) y retorna(e) funciones que permiten girar un elemento 360 grados y 0 grados en su eje Y por un segundo.

```

14
15 function cambio(e){
16     var img=e.getAttribute("data-social");
17     e.src="imgs/"+img+"1"+".jpg";
18 }
19
20 function regresa(e){
21     var img=e.getAttribute("data-social");
22     e.src="imgs/"+img+".jpg";
23 }
24
25 function gira(e){
26     e.style.transform="rotateY(360deg)";
27     e.style.transition="all 1s";
28 }
29
30 function retorna(e){
31     e.style.transform="rotateY(0deg)";
32     e.style.transition="all 1s";
33 }
34

```

Figura 153 : Caso práctico

Fuente.- Elaboración Propia

3. A continuación, en el archivo programa.js, enlazamos a los elementos de las páginas, los eventos correspondientes, tal como se muestra.

```

29
30 > function retorna(e){...
31 }
32
33 /*eventos*/
34 document.body.setAttribute("onload","carrousel()");
35
36 for(var i=0; i<document.getElementsByClassName("social").length; i++) {
37     document.getElementsByClassName("social")[i].setAttribute("onmouseover", "cambio(this)");
38     document.getElementsByClassName("social")[i].setAttribute("onmouseout", "regresa(this)");
39 }
40
41 for(var i=0; i<document.getElementsByClassName("imgfooter").length;i++){
42     document.getElementsByClassName("imgfooter")[i].setAttribute("onmouseover", "gira(this)");
43     document.getElementsByClassName("imgfooter")[i].setAttribute("onmouseout", "retorna(this)");
44 }
45
46 /*fin de eventos*/
47

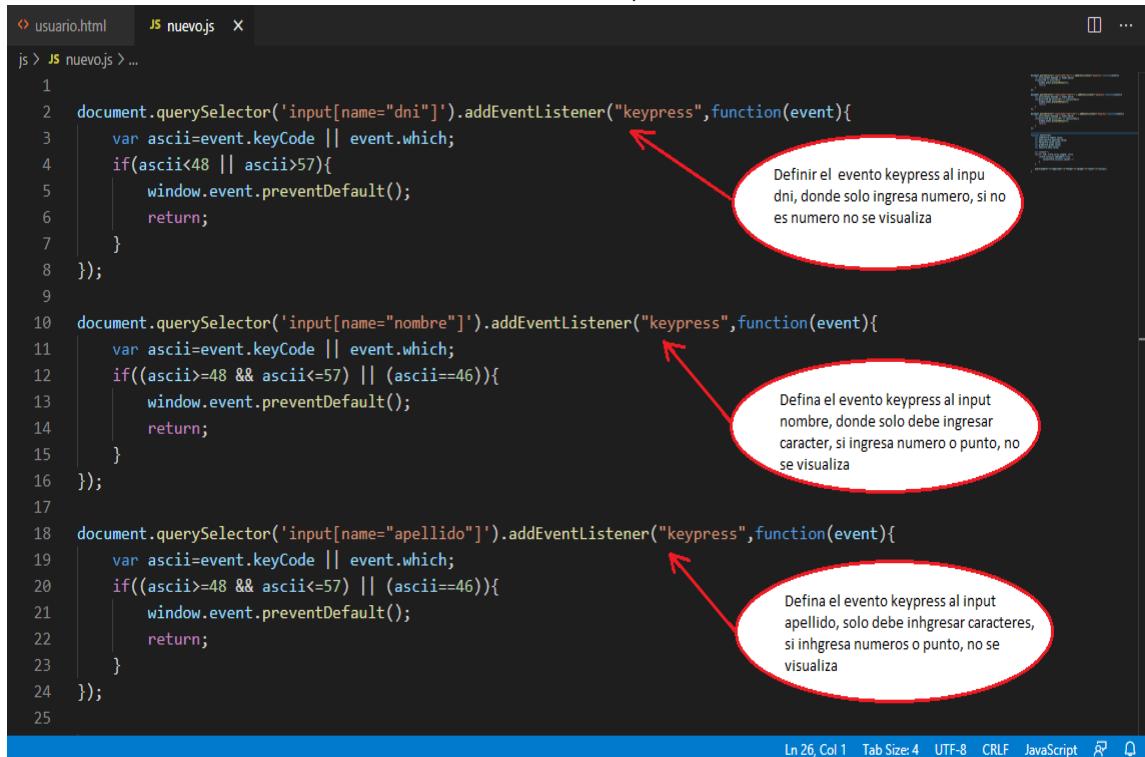
```

Figura 154: Caso práctico

Fuente.- Elaboración Propia

### Definiendo la programación: nuevo.js.

- En el archivo nuevo.js, defina el evento keypress a los <input>: nombre DNI y apellido; donde: el DNI solo debe ingresar números, y el nombre y apellido solo debe ingresar caracteres. Defina la función asociada al evento, tal como se muestra.



```

1  document.querySelector('input[name="dni"]').addEventListener("keypress",function(event){
2      var ascii=event.keyCode || event.which;
3      if(ascii<48 || ascii>57){
4          window.event.preventDefault();
5          return;
6      }
7  });
8 });

9
10 document.querySelector('input[name="nombre"]').addEventListener("keypress",function(event){
11     var ascii=event.keyCode || event.which;
12     if((ascii>=48 && ascii<=57) || (ascii==46)){
13         window.event.preventDefault();
14         return;
15     }
16 });
17
18 document.querySelector('input[name="apellido"]').addEventListener("keypress",function(event){
19     var ascii=event.keyCode || event.which;
20     if((ascii>=48 && ascii<=57) || (ascii==46)){
21         window.event.preventDefault();
22         return;
23     }
24 });
25

```

Ln 26, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️

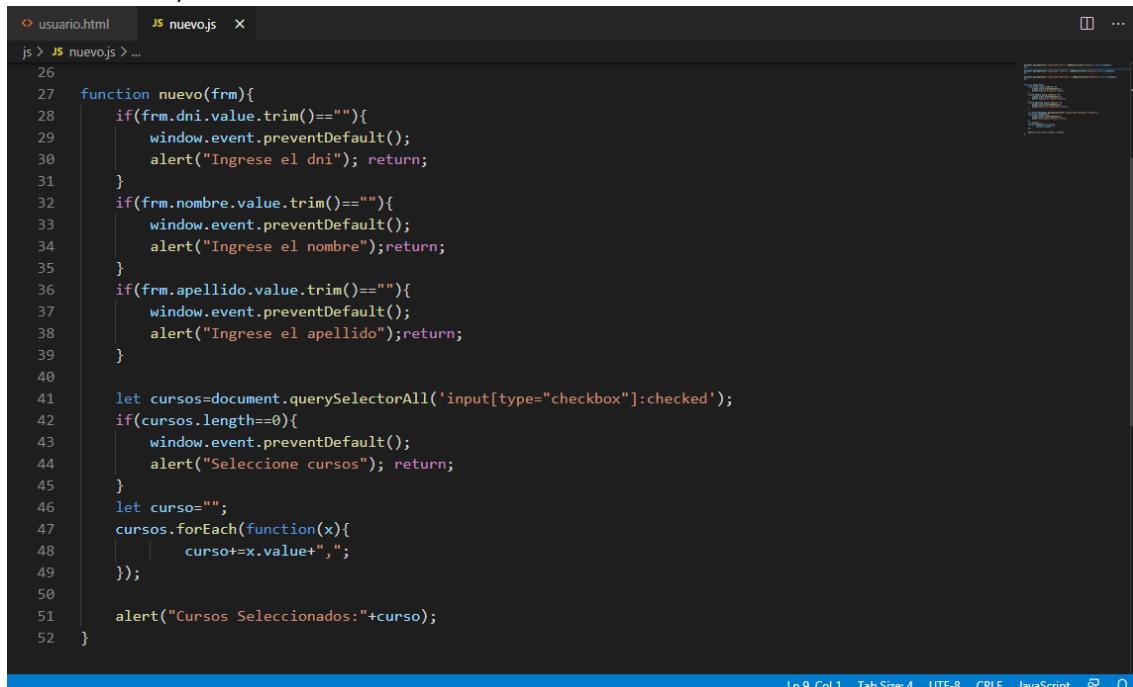
Three callout bubbles with arrows pointing to the code blocks:

- Top bubble: "Definir el evento keypress al input dni, donde solo ingresa numero, si no es numero no se visualiza"
- Middle bubble: "Definir el evento keypress al input nombre, donde solo debe ingresar caracter, si ingresa numero o punto, no se visualiza"
- Bottom bubble: "Definir el evento keypress al input apellido, solo debe ingresar caracteres, si ingresa numeros o punto, no se visualiza"

Figura 155: Caso práctico

Fuente.- Elaboración Propia

- Defina del método nuevo, donde verifica los datos ingresados, donde si están todos correctos, visualizamos los cursos seleccionados.



```

26
27 function nuevo(frm){
28     if(frm.dni.value.trim()!=""){
29         window.event.preventDefault();
30         alert("Ingrese el dni"); return;
31     }
32     if(frm.nombre.value.trim()!=""){
33         window.event.preventDefault();
34         alert("Ingrese el nombre"); return;
35     }
36     if(frm.apellido.value.trim()!=""){
37         window.event.preventDefault();
38         alert("Ingrese el apellido"); return;
39     }
40
41     let cursos=document.querySelectorAll('input[type="checkbox"]:checked');
42     if(cursos.length==0){
43         window.event.preventDefault();
44         alert("Seleccione cursos"); return;
45     }
46     let curso="";
47     cursos.forEach(function(x){
48         | | | curso+=x.value+",";
49     });
50
51     alert("Cursos Seleccionados:"+curso);
52 }

```

Ln 9, Col 1 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️

Figura 156 : Caso práctico

Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña nuevo.html, y presiona la tecla F5.

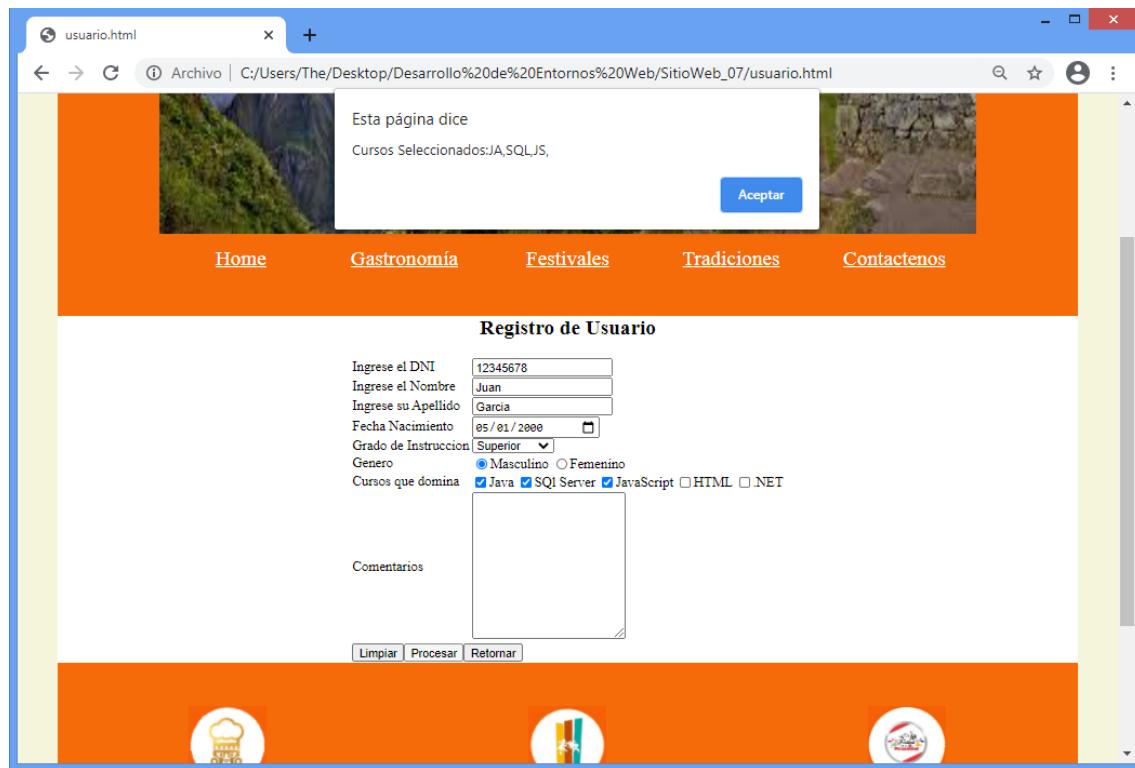


Figura 157: Caso práctico  
Fuente.- Elaboración Propia

# Resumen

1. Los formularios en JavaScript nos permiten mucho más que recopilar datos para enviarlos a una dirección o programa de tratamiento.
2. El objeto Form es un subobjeto del objeto document y este a su vez, lo es del objeto Window. Así como para crear una página en HTML se utilizan las etiquetas <HTML> Y </HTML>, lo mismo sucede con un formulario: el formulario debe estar contenido entre las etiquetas <form> y </form>.
3. Las propiedades del objeto form: name, method, action, target.
4. Los métodos del objeto form son submit, reset.
5. Las propiedades de los elementos de un formulario son: type, form, name, value.
6. Los eventos más utilizados en el manejo de formularios: onclick, onchange, onfocus, onblur.
7. Cuando se carga una página web, el navegador crea automáticamente un array llamado forms y que contiene la referencia a todos los formularios de la página.
8. Para acceder a un formulario, se realiza a través de su nombre (atributo name) o a través de su atributo id. El objeto document permite acceder directamente a cualquier formulario mediante su atributo name.

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://desarrolloweb.com/articulos/910.php>
- <https://uniwebsidad.com/libros/JavaScript/capitulo-7>
- [https://aprende-web.net/JavaScript/js7\\_1.php](https://aprende-web.net/JavaScript/js7_1.php)
- <https://uniwebsidad.com/libros/JavaScript/capitulo-7/utilidades-basicas-para-formularios>
- <https://www.arkaitzgarro.com/JavaScript/capitulo-16.html>
- <https://desarrolloweb.com/manuales/50>

## 5.2. EXPRESIONES REGULARES

### 5.2.1. Introducción

La búsqueda de caracteres o secuencias concretas de caracteres en documentos forma parte de las tareas estándar y recurrentes de la tecnología de la información. Por lo general, el objetivo es modificar o sustituir fragmentos de texto o líneas de código, cuya complejidad aumenta en función de las veces que la secuencia de caracteres aparece en el documento. En la década de 1950 se encontró una solución basada en las lenguas formales de la informática teórica que sigue presente en el desarrollo actual de software y que permite simplificar estas tareas repetitivas mediante el uso de las denominadas expresiones regulares (en inglés, regular expressions).

### 5.2.2. Definición de expresiones regulares

Las expresiones regulares son secuencia de caracteres que forman un patrón para encontrar o reemplazar una determinada combinación de caracteres dentro de una cadena de texto.

El encontrar una cadena o reemplazar una cadena por otra puede hacerse usando los métodos del objeto String, pero el problema surge cuando no tenemos una subcadena fija y concreta, sino que queremos buscar un texto que responda a un cierto esquema, como, por ejemplo: buscar aquellas palabras que comienzan con http: y finalizan con una \, o buscar palabras que contengan una serie de números consecutivos, etc.; es en estos casos cuando tenemos que utilizar las expresiones regulares.

En JavaScript, las expresiones regulares también son objetos. Estos patrones son utilizados a través de los métodos exec y test del objeto RegExp, así como los métodos match, replace, search y split del objeto String.

#### 5.2.2.1. Creación de expresiones

La sintaxis básica para crear expresiones regulares es la siguiente:

/patrón/modificadores;

Por ejemplo:

```
var patt = /w3schools/i
```

Donde:

/w3schools/i es la expresión regular.

w3schools es el patrón para ser usado en una búsqueda.

i es el modificador insensitive que significa que la búsqueda no considera mayúsculas o minúsculas.

Una expresión regular puede crearse de cualquiera de las siguientes dos maneras:

#### 1. Utilizando una representación literal de la expresión:

```
var re = /ab+c/;
```

La representación literal compila la expresión regular una vez que el script ha terminado de cargar. Es recomendable utilizar esta representación cuando la expresión regular permanecerá sin cambios durante la ejecución del script, puesto que ofrece un mejor desempeño.

### **Constructor del objeto RegExp:**

```
var re = new RegExp("ab+c");
```

El uso del constructor ofrece una compilación de la expresión regular en tiempo de ejecución. Su uso es recomendable en aquellos escenarios en que el patrón de la expresión regular pueda cambiar durante la ejecución del script, o bien, se desconoce el patrón, dado que se obtiene desde otra fuente, cuando es suministrado por el usuario, por ejemplo.

### **Modificadores**

Los modificadores son usados para realizar búsquedas case-insensitive y en toda la cadena:

Modificador	Descripción
i	Realiza comparaciones case-insensitive
g	Realiza una búsqueda global (encuentra todas las coincidencias en lugar de detenerse después de encontrar la primera)
m	Realiza comparaciones en todas las líneas

### **Corchetes**

Los corchetes se usan para buscar un rango de caracteres:

Expresión	Descripción
[abc]	Busca cualquier carácter indicado entre los corchetes
[^abc]	Busca cualquier carácter que no se encuentra entre los corchetes
[0-9]	Busca cualquier carácter dígito
[^0-9]	Busca cualquier carácter que no sea dígito
(x y)	Busca cualquier carácter de las alternativas especificadas

En la tabla que sigue se muestran los caracteres comodín usado para crear los patrones y su significado, junto a un pequeño ejemplo.

Significado	Ejemplo	Resultado
\ Marca de carácter especial	/\\$ftp/	Busca la palabra \$ftp
^ Comienzo de una línea	/^-/	Líneas que comienzan por -
\$ Final de una línea	/s\$/	Líneas que terminan por s
. Cualquier carácter (menos salto de línea)	/\b.\b/	Palabras de una sola letra
Indica opciones	/(L l f )ocal/	Busca Local, local, focal
( ) Agrupar caracteres	/(vocal)/	Busca vocal
[ ] Conjunto de caracteres optionales	/escrib[aoe]/	Vale escriba, escribo, escribe

La tabla que sigue describe los modificadores que pueden usarse con los caracteres que forman el patrón. Cada modificador actúa sobre el carácter o el paréntesis inmediatamente anterior.

Descripción		Ejemplo	Resultado
*	Repetir 0 o más veces	/l*234/	Valen 234, 1234, 11234...
+	Repetir 1 o más veces	/a*mar/	Valen amar, aamar, aaamar...
?	1 o 0 veces	/a?mar/	Valen amar, mar.
{n}	Exactamente n veces	/p{2}sado/	Vale ppsado
{n,}	Al menos n veces	/(m){2}ala/	Vale mmala, mmmala....
{m,n}	entre m y n veces	/tal{1,3}a/	Vale tala, talla, tallia

Los siguientes son caracteres especiales o metacaracteres para indicar caracteres de texto no imprimibles, como puedan ser el fin de línea o un tabulador, o grupos predefinidos de caracteres (alfabéticos, numéricos, etc...)

Significado		Ejemplos	Resultado
\b	Principio o final de palabra	/\bver\b/	Encuentra ver en "verde", pero no en "verde"
\B	Frontera entre no-palabras	/\Bver\B/	Empareja ver con "Valverde" pero no con "verde"

\d	Un dígito	/[A-Z]\d/	No falla en "A4"
\D	Alfabético (no dígito)	/[A-Z]\D/	Fallaría en "A4"
\O	Carácter nulo		
\t	Carácter ASCII 9 (tabulador)		
\f	Salto de página		
\n	Salto de línea		
\w	Cualquier alfanumérico, [a-zA-Z0-9_ ]	/\w+ /	Encuentra <i>frase</i> en " <i>frase.</i> ", pero no el . (punto).
\W	Opuesto a \w ([a-zA-Z0-9_ ])	/\W/	Hallaría sólo el punto (.)
\s	Carácter tipo espacio (como tab)	/\sSi\s/	Encuentra <i>Si</i> en " <i>Digo Si</i> ", pero no en " <i>Digo Siéntate</i> "
\S	Opuesto a \s		
\cX	Carácter de control X	\c9	El tabulador
\oNN	Carácter octal NN		
\xhh	El hexadecimal hh	/\x41/	Encuentra la A (ASCII Hex41) en " <i>letra A</i> "

### 5.2.2.2. Manejo de caracteres especiales

Una expresión regular se construye con una cadena de texto que representa el formato que debe cumplir el texto. En JavaScript se puede hacer de dos formas, bien instanciando una clase RegExp pasando como parámetro la cadena de formato, bien poniendo directamente la cadena de formato, en vez de entre comillas, entre /

```
// Son equivalentes
varreg = newRegExp("aa")
varreg = /aa/
```

Ver si un texto tiene una determinada secuencia de letras o números fija es fácil. La expresión simplemente es esa cadena y podemos ver si un texto la tiene usando el método match() de ese texto

```

varreg = /JavaScript/;
"holo JavaScript".match(reg);

// devuelve un array de 1 elemento ["JavaScript"], indicando que sí existe esa
cadena dentro del texto

"adios tu".match(reg);

// devuelve null, indicando que no existe JavaScript dentro de "adios tu".

```

No es necesario definir la expresión regular antes, podemos hacerlo así

```

"holo JavaScript".match(/JavaScript/);

// Devuelve ["JavaScript"]

```

Y para verificar si existe o no la cadena, podemos poner directamente un if

```

if("holo JavaScript".match(/JavaScript/)
{
    // Pasará por aquí, porque un array con un elemento se evalúa como true en
el if
}

if("adios tu".match(/JavaScript/) {
    // No pasa por aquí, null se evalúa como false en el if.
}

```

### **Caracteres no alfabéticos ni numéricos**

Algunos de los caracteres no numéricos ni alfabéticos tienen un significado especial (lo vemos más adelante), como por ejemplo [ ] { } ( ) \* . ^ \$ etc. No podemos ponerlos tal cual si forman parte de nuestro formato, debemos "escaparlos" poniendo \ delante

```

"esto es un *".match(/\*/);
// Devuelve ["*"] indicando que existe un asterisco.

```

### **Conjunto opcional de caracteres**

A veces nos da igual que una letra, por ejemplo, sea mayúscula o minúscula, o queremos que sea una vocal, o un dígito. Cuando queremos que una de las letras de nuestro texto pueda ser una cualquiera de un conjunto de letras determinado, las ponemos entre [] en nuestra expresión. Por ejemplo, si nos vale "JavaScript" y "JavaScript", podemos poner la expresión como /[Jj]avascript/ para indicar que nos vale J mayúscula o j minúscula

```
"JavaScript con minuscula".match(/[Jj]avascript/);  
// Sí encuentra la cadena  
  
"JavaScript con minuscula".match(/[Jj]avascript/);  
// También la encuentra.
```

Si los caracteres válidos son varios y van ordenados según el juego de caracteres, podemos poner el primero y el último separado por un -. Por ejemplo, [a-z] vale para cualquier letra minúscula, [0-9] para cualquier dígito y [a-zA-Z] para cualquier letra mayúscula o minúscula

```
"aa2bb".match(/[0-9]/); // Encuentra el 2, devolviendo ["2"].
```

Podemos hacer lo contrario, es decir, que la letra no esté en ese conjunto de caracteres. Se hace poniendo el juego de caracteres que no queremos entre [^ y ]. Por ejemplo, para no dígitos pondríamos [^0-9]

```
"22 33".match(/\^0-9/); // Encuentra el espacio en blanco, devolviendo [" "]
```

### Conjuntos habituales

Hay varios conjuntos que se usan con frecuencia, como el de letras [a-zA-Z], el de dígitos [0-9] o el de espacios en blanco (espacio, tabulador, etc.). Para estos conjuntos la expresión regular define formas abreviadas, como

\w	para letras, equivalente a [a-zA-Z]
\W	para no letras, equivalente a [^a-zA-Z]
\d	para dígitos, equivalente a [0-9]
\D	para no dígitos, equivalente a [^0-9]
\s	para espacios en blanco (espacios, tabuladores, etc.).
\S	para no espacios en blanco.

Por ejemplo

```
"aa2bb".match(/\d/); // Encuentra el 2, devolviendo ["2"]
```

### Repetición de caracteres

Podemos querer buscar por ejemplo un conjunto de tres dígitos, podemos hacerlo repitiendo tres veces el \d

```
"aa123bb".match(/\d\d\d/); // Encuentra el 123, devolviendo ["123"]
```

Pero esta forma es un poco engorrosa si hay muchos caracteres y es poco versátil. Las expresiones regulares nos permiten poner entre {} un rango de veces que debe repetirse. Por ejemplo

/\d{3}/	Busca 3 dígitos en la cadena
/\d{1,5}/	Busca entre 1 y 5 dígitos en la cadena.
/\d{2,}/	Busca 2 dígitos o más en la cadena.

Aplicando estas expresiones regulares tenemos:

```
"1234".match(/\d{2}/);
["12"]

"1234".match(/\d{1,3}/);
["123"]

"1234".match(/\d{3,10}/)
["1234"]
```

También suele haber rangos habituales como 0 o más veces, 1 o más veces, 0 ó 1 vez. Estos rangos habituales tienen caracteres especiales que nos permiten ponerlos de forma más simple.

- \* equivale a 0 o más veces {0,}
- + equivale a 1 o más veces {1,}
- ? equivale a 0 ó 1 vez {0,1}

Por ejemplo

```
"a2a".match(/a\d+a/); // Encuentra a2a
"a234a".match(/a\d+a/); // Encuentra a234a
```

Cosas como \* o + encuentran el máximo posible de caracteres. Por ejemplo, si nuestro patrón es /a+/ y nuestra cadena es "aaaaa", el resultado será toda la cadena

```
"aaaaa".match(/a+/); // Devuelve ["aaaaa"]
```

¿Para hacer que se encuentre lo menos posible, se pone un ? detrás. Así, por ejemplo, si nuestra expresión regular es /a+?/ y nuestra cadena es "aaaaa", sólo se encontrará una "a"

```
"aaaaa".match(/a+?/); // Devuelve ["aaaaa"]
```

El comportamiento inicial se conoce como "greedy" o codicioso, en el que el patrón intenta coger la mayor parte de la cadena de texto posible. El segundo comportamiento se conoce como "nongreedy" o no codicioso, en el que el patrón coge lo menos posible de la cadena.

### Extraer partes de la cadena

A veces nos interesa no sólo saber si una cadena cumple un determinado patrón, sino extraer determinadas partes de él. Por ejemplo, si una fecha está en el formato "27/11/2012" puede interesarnos extraer los números. Una expresión regular que vale para esta cadena puede ser:

```
/\d{1,2}\/\d{1,2}\/\d{4}/
```

Suponiendo que el día y el mes puedan tener una cifra y que el año sea obligatoriamente de 4 cifras. En este caso:

```
"27/11/2012".match(/\d{1,2}\/\d{1,2}\/\d{4}/);
```

Nos devuelve un array con un único elemento que es la cadena "27/11/2012". Para extraer los trozos, únicamente debemos poner entre paréntesis en la expresión regular aquellas partes que nos interesan. Es decir,

```
/(\d{1,2})\/(\d{1,2})\/(\d{4})/
```

Si ahora ejecutamos el método `match()` con la misma cadena anterior, obtendremos un array de 4 cadenas. La primera es la cadena completa que cumple la expresión regular. Los otros tres elementos son lo que cumple cada uno de los paréntesis

```
"27/11/2012".match(/(\d{1,2})\/(\d{1,2})\/(\d{4})/); // Devuelve el array  
["27/11/2012", "27", "11", "2012"]
```

Los paréntesis también nos sirven para agrupar un trozo y poner detrás uno de los símbolos de cantidades. Por ejemplo

```
"xyxyxyxy".match/(xy)+/; // Se cumple, hay xy una o más veces.
```

### Usar lo encontrado en la expresión

Las partes de la cadena que cumplen la parte de expresión regular entre paréntesis se pueden reutilizar en la misma expresión regular. Estas partes encontradas se van almacenando en \1, \2, \3... y podemos usarlas. Esta posibilidad es realmente interesante si queremos, por ejemplo, verificar que una cadena de texto va cerrada entre comillas del mismo tipo, es decir, queremos buscar algo como 'esto' o "esto", pero no nos vale 'esto'.

La expresión regular para buscar una cadena entre este tipo de comillas puede ser `/(["").*\1/` es decir, buscamos una ' o una " con ["]. Hacemos que lo que se encuentre se guarde metiéndolo entre paréntesis (["]) y a partir de ahí nos vale cualquier conjunto de caracteres terminados en \1, que es lo que habíamos encontrado al principio.

```
"'hola tu' tururú".match(/(["").*\1/); // Devuelve ["'hola tu'", """]
"\\"hola tu' tururú".match(/(["").*\1/); // Devuelve null, la cadena comienza con "y
termina en '
```

### Ignorar lo encontrado

A veces nos interesa encontrar una secuencia que se repita varias veces seguidas y la forma de hacerlo es con los paréntesis, por ejemplo, si ponemos `/(pa){2}/` estamos buscando "papa". Para evitar que esos paréntesis guarden lo encontrado en \1, podemos poner ?:, tal que así `/(?:pa){2}/`, de esta forma encontraremos "papa", pero se nos devolverá el trozo "pa" encontrado ni lo tendremos disponible en \1. Compara las dos siguientes:

```
"papa".match(/(pa){2}/); // Devuelve ["papa", "pa"]
"papa".match(/(?:pa){2}/); // Devuelve ["papa"]
```

### Posición de la expresión

A veces nos interesa que la cadena busque en determinadas posiciones. Las expresiones regulares nos ofrecen algunos caracteres espaciales para esto.

^ indica el principio de cadena, por ejemplo, `/^hola/` vale si la cadena "hola" está al principio

```
"hola tu".match(/^hola/); // Devuelve ["hola"]
"pues hola tu".match(/^hola/); // Devuelve null
```

final de la cadena, por ejemplo `/tu$/` vale si la cadena termina en "tu"

```
"hola tu".match(/tu$/); // Devuelve ["tu"]
"hola tu tururú".match(/tu$/); // Devuelve null
```

\b indica una frontera de palabra, es decir, entre un carácter "letra" y cualquier otra cosa como espacios, fin o principio de línea, etc. De esta forma, por ejemplo, /\bjava\b/ buscará la palabra java, pero ignorará JavaScript

```
"java es güay".match(/\bjava\b/); // Devuelve ["java"]
"JavaScript es güay".match(/\bjava\b/); // Devuelve null
```

\B es lo contrario de \b, así, por ejemplo, /\bjava\B/ buscará una palabra que empiece por "java", pero no sea sólo java, sino que tenga algo más

```
"java es güay".match(/\bjava\B/); // Devuelve null
"JavaScript es güay".match(/\bjava\B/); // Devuelve ["java"]
```

(?=expresion) sirve para posicionar el resto de la expresión regular y buscar antes o después. Por ejemplo, si queremos buscar un número que vaya delante de km, podemos hacer esto

/\d+(?= km)/

Es decir, uno o más dígitos seguidos de un espacio y las letras km. La diferencia con esta expresión (/d+ km/) es que en el primer caso sólo casan con la expresión los números, mientras que en el segundo caso se incluye también el " km"

```
"11 millas 10 km".match(/\d+(?= km)/); // Devuelve ["10"]
"11 millas 10 km".match(/\d+ km/); // Devuelve ["10 km"]
```

Hay que tener cuidado si buscamos detrás, ¿porque como el trozo (?=expresion) no se tiene en cuenta, sigue contando para el resto de la expresión. Por ejemplo, si queremos extraer la parte decimal de "11.22" podríamos pensar en hacer esto

/(?=\.).\d+/  
/

Pero no funciona porque él . (punto) decimal no se "consume" con (?=\.), así que debemos tenerlo en cuenta y ponerlo detrás, así

/(?=\.).\.\d+/  
/

Por ejemplo:

```
"11.22".match(/(=?\.)\d+/); // Devuelve null
"11.22".match(/(=?\.)\.\d+/); // Devuelve [".22"]
```

(?!expresion) hace lo contrario que (=expresion), es decir, busca una posición donde no se cumpla expresión. Por ejemplo, para sacar lo que no sean km de "11 km, 12 km, 14 m" podemos poner

```
\d{2}(?! km)/
```

Por ejemplo:

```
"11 km 12 km 14 m".match(/\d{2}(?! km)/); // Devuelve ["14"]
```

### Flags de opciones

Hemos visto que una expresión regular es /expresion/. Podemos poner algunos flags detrás, básicamente unas letras que cambian algo el comportamiento

- i es para ignorar mayúsculas y minúsculas

```
"hola".match(/HOLA/); // Devuelve null
"hola".match(/HOLA/i); // Devuelve ["hola"]
```

- g es para buscar todas las veces posibles la expresión, no sólo una vez

```
"11 223 44 66 77".match(/\d+/); // Devuelve ["11"]
"11 223 44 66 77".match(/\d+/g); // Devuelve ["11", "223", "44", "66", "77"]
```

- m busca en cadenas con retornos de carro \n considerando estos como inicios de línea ^ o fin \$

```
"hola\ntu".match(/^tu/); // Devuelve null
"hola\ntu".match(/^tu/m); // Devuelve ["tu"]
"hola\ntu".match(/hola$/); // Devuelve null
"hola\ntu".match(/hola$/m); // Devuelve ["hola"]
```

### Otros métodos de cadena y de expresión regular

Para todos estos ejemplos hemos usado el método match() de la clase String, ya que nos devuelve un array con las cosas que se encuentran y viendo los resultados es la forma más clara de ver cómo funcionan las distintas partes de la expresión regular. Sin embargo, tanto String como RegExp tienen otros métodos útiles

**String.search(/expresión/)**

Devuelve la posición donde se encuentra esa expresión dentro de la cadena, o -1 si no se encuentra.

**String.replace(/expresión/,cadena)**

Busca el trozo de cadena que casa con la expresión y la reemplaza con lo que le pasemos en el parámetro cadena. Este método tiene además un detalle interesante. Cuando en la expresión regular tenemos paréntesis para extraer algunas partes de la cadena, la misma expresión regular recuerda qué ha encontrado. En el método replace, si en la cadena de segundo parámetro aparecen cosas como \$1, \$2, utilizará lo encontrado.

```
"ho3la".replace(/\d/,"X"); // Devuelve "hoXla"  
"ho3la".replace(/(\d)/,"-$1-"); // Devuelve "ho-3-la"
```

**String.split(/expresión/)**

Usa lo que sea que case con la expresión como separador y devuelve un array con la cadena partida en trozos por ese separador

```
"hola, dos tres; cuatro".split(/\W+/); // Devuelve ["hola", "dos", "tres", "cuatro"]
```

**RegExp constructor**

Además de crear las expresiones regulares con /expresión flags, podemos hacerlo con un new de la clase RegExp, por ejemplo, new RegExp("expresión","flags").

```
varreg = new RegExp("\\d+","g");  
  
"11 22 33".match(reg); // Devuelve ["11","22","33"]
```

Hay que fijarse en este caso que las \ debemos escaparlas, con \\

**RegExp.exec()**

Es similar a match() de String, pero sólo devuelve un resultado y hace que RegExp guarde la posición en la que lo ha encontrado. Sucesivas llamadas a exec(), nos irán devolviendo los siguientes resultados

```

varreg = newRegExp("\\d+");

reg.exec("11 22 33"); // Devuelve ["11"]
reg.exec("11 22 33"); // Vuelve a devolver ["11"], puesto que no hay flag g

varreg = newRegExp("\\d+","g");

reg.exec("11 22 33"); // Devuelve ["11"]
reg.exec("11 22 33"); // Vuelve a devolver ["22"], puesto que si hay flag g
reg.exec("11 22 33"); // Vuelve a devolver ["33"], puesto que si hay flag g
reg.exec("11 22 33"); // Devuelve null, ya no hay más resultados.
reg.exec("11 22 33"); // Vuelve a devolver ["11"], después de devolver null
la RegExp se "reinicializa"

```

### RegExp.test()

Similar a exec(), pero en vez de devolver lo encontrado, devuelve true si ha encontrado algo o false si no. Como la expresión regular recuerda las búsquedas anteriores, sucesivas llamadas a test() pueden devolver resultados distintos

```

varreg = newRegExp("\\d+","g");

reg.test("11 22 33"); // Devuelve true, porque encuentra el ["11"]
reg.test("11 22 33"); // Vuelve a devolver true, porque encuentra el ["22"], puesto
que si hay flag g
reg.test("11 22 33"); // Vuelve a devolver true, porque encuentra el ["33"], puesto
que si hay flag g
reg.test("11 22 33"); // Devuelve false, ya no hay más resultados.
reg.test("11 22 33"); // Vuelve a devolver true, porque vuelve a encontrar el ["11"],
después de devolver null la RegExp se "reinicializa"

```

### Ejemplo: Probar si existe una cadena

Vamos a comprobar si una cadena está contenida en otra cadena. Utilizar una expresión regular JavaScript para definir un patrón de búsqueda, y luego aplicar el patrón contra de la cadena a buscar, mediante el método RegExp. Haremos que coincida con cualquiera cadena que tenga las dos palabras Cook y Book en ese orden:

Solución.

```

var cookbookString = new Array();
cookbookString[0] = "Joe's Cooking Book";
cookbookString[1] = "Sam's Cookbook";
cookbookString[2] = "JavaScript CookBook";
cookbookString[3] = "JavaScript BookCook";

// patrón de búsqueda
var pattern = /Cook.*Book/;
for (var i = 0; i < cookbookString.length; i++)

```

```
alert(cookbookString[i] + " " + pattern.test(cookbookString[i]));
```

La primera y tercera cadenas tienen un resultado positivo, mientras que la segundo y cuarta no. El método de prueba RegExp toma dos parámetros: la cadena de prueba, y un modificador opcional. Se aplica la expresión regular con la cadena y devuelve true si hay una coincidencia, falso si no hay.

En el ejemplo, el patrón es la palabra Cook y la palabra Book, que aparecen en algún lugar en la cadena, la palabra Book después de la palabra Cook. Puede haber el número de caracteres que sea entre las dos palabras, incluyendo que no haya nada indicados por los caracteres especiales, punto(.) y el asterisco(\*)

El decimal en las expresiones regulares es un carácter especial que coincide con cualquier carácter excepto el carácter de nueva línea. En el patrón de ejemplo, el decimal es seguido por un asterisco, que coincide con el carácter anterior cero o más veces. Combinados, generan un patrón de coincidencia con cero o más de cualquier carácter, excepto una línea nueva.

### **Ejemplo: Encontrar y destacar todas las instancias de un patrón**

Quieres encontrar todas las instancias de un patrón dentro de una cadena. Para ello hay que utilizar el método exec en la expresión regular con el flag (g).

Vamos a decir como ejemplo cualquier palabra que empieza con t y termina con he y nos dará igual el número de caracteres entre ellos.

```
var searchString = "Now is the time and this is the time and that is the time";
var pattern = /tw*e/g;
var matchArray;
var str = "";
while((matchArray = pattern.exec(searchString)) != null) {
  str+="at " + matchArray.index + " we found " + matchArray[0] + "";
}
document.getElementById("results").innerHTML=str;
```

El método exec de la expresion regular, devuelve null si no se encuentra la coincidencia, o un array con la información encontrada. En el array se encuentra el valor actual que ha encontrado, el índice de la cadena donde se encuentra la coincidencia, cualquier coincidencia de subcademas entre paréntesis, y la cadena original.

### **Ejemplo: Reemplazar un patrón por una nueva cadena**

Para ello hay que usar el método replace del objeto String con una expresión regular.

```
var searchString = "Now is the time, this is the time";
var re = /tw{2}e/g;
var replacement = searchString.replace(re, "place");
alert(replacement); // Now is the place, this is the place
```

### **Ejemplo: Intercambio de palabras en una cadena usando paréntesis de captura**

Hay que usar paréntesis de captura y una expresión regular para encontrar y recordar los dos nombres de la cadena, y revertirlos.

En el ejemplo vamos a intercambiar el nombre de orden, poniendo el nombre al final y el apellido primero.

```
var name = "Pedro Ventura";
var re = /^(w+)s(w+)$/;
var newname = name.replace(re,"$2, $1");
```

Los paréntesis de captura nos permitirán no sólo para que coincida con los patrones preestablecidos en una cadena, sino que además hace referencia al subcadenas coincidentes. Las cadenas coincidentes son referenciadas numéricamente de izquierda a derecha y son representadas con el uso de "\$1" y "\$2" en el método replace de String.

Como las dos palabras están separadas por un espacio, es fácil de crear la expresión regular, que recoge el nombre (las dos palabras), y el nombre será accesible usando "\$1" y el apellido con "\$2".

## LABORATORIO 1

### Trabajando con expresiones regulares

En este laboratorio vamos a crear una página HTML llamada registro.html para ingresar los datos del usuario. En este proceso realice las siguientes operaciones:

- a. Validar que el DNI tenga 8 dígitos
- b. Validar que el nombre y apellido no estén vacíos
- c. Validar que el fono sea numérico entre 9 a 15 dígitos
- d. Validar que el email tenga el formato de correo electrónico.

### 1. Creando el Sitio Web

Defina una carpeta llamado SitioWeb\_07, en ella crea las siguientes subcarpetas, tal como se muestra.

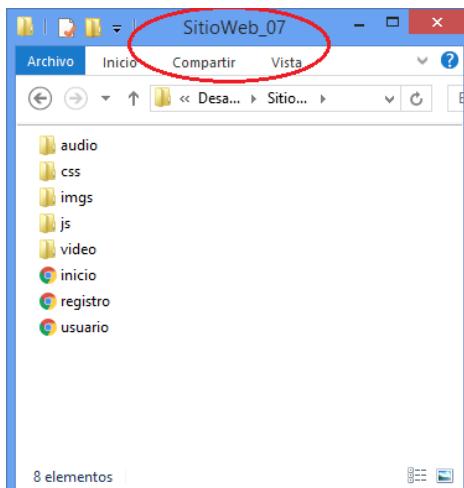


Figura 158: Caso práctico  
Fuente.- Elaboración Propia

### 2. Trabajando con Visual Studio Code

Para crear una página web, abrir el aplicativo Visual Studio Code.

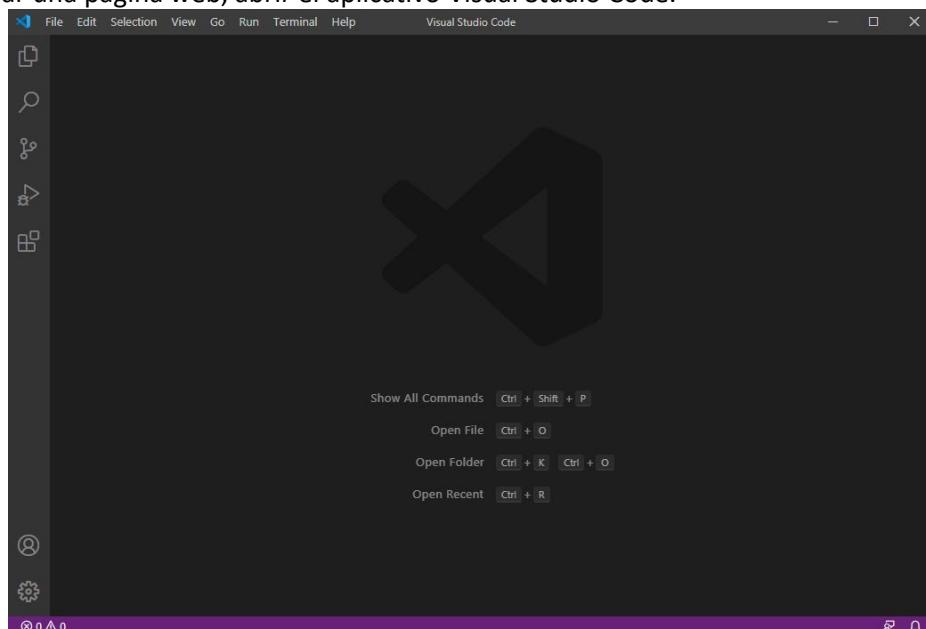


Figura 159 : Caso práctico  
Fuente.- Elaboración Propia

Para trabajar con la carpeta llamada SitioWeb\_07 en Visual Studio Code, arrastre la carpeta hacia la ventana del Visual, tal como se muestra; y luego soltarla.

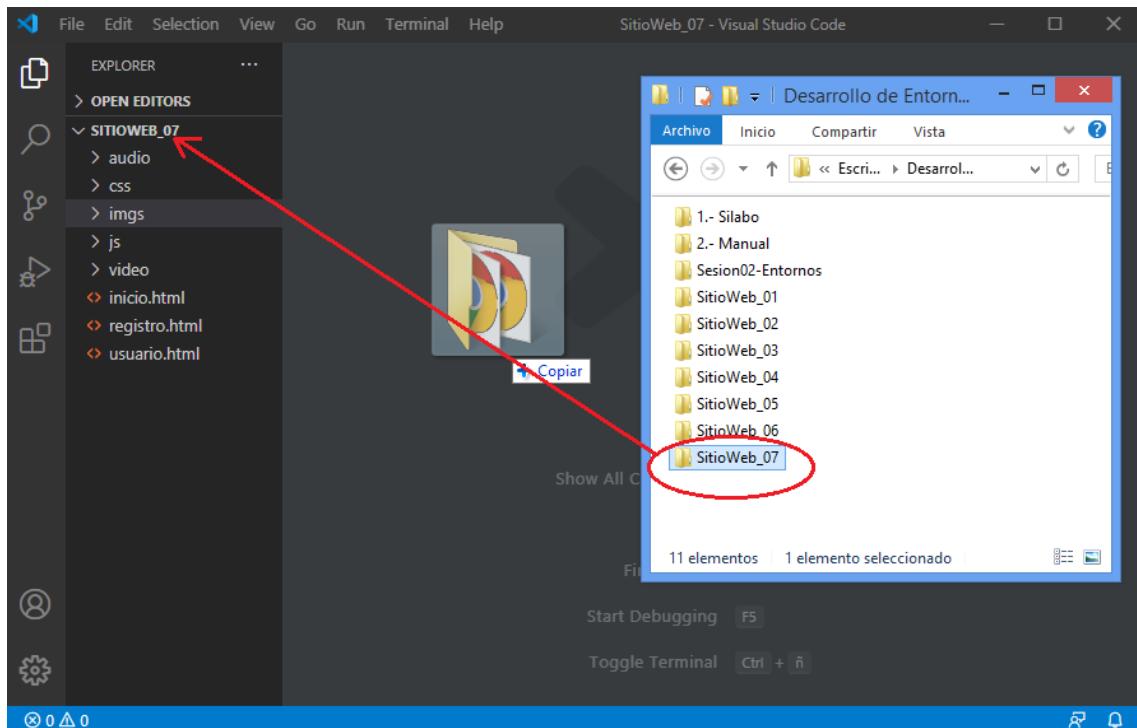


Figura 160 : Caso práctico  
Fuente.- Elaboración Propia

Donde se visualiza la carpeta en la parte izquierda de la ventana del aplicativo

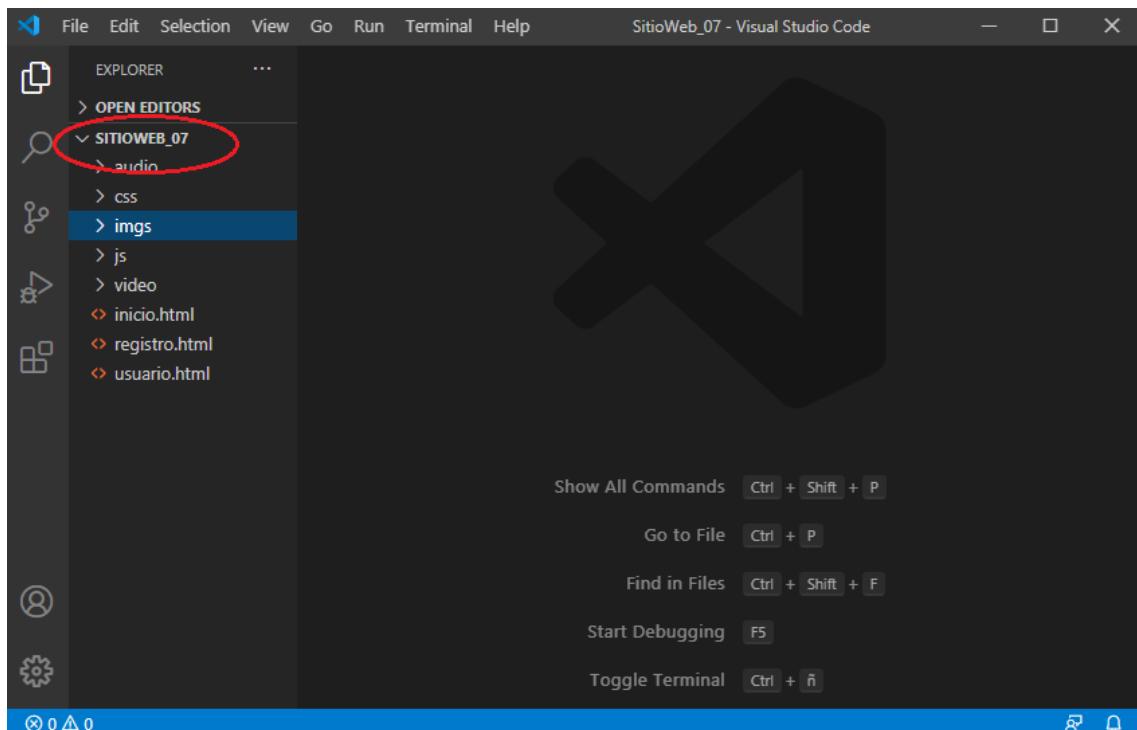


Figura 161: Caso práctico  
Fuente.- Elaboración Propia

## 1. Diseñando la página web

En esta etapa vamos a diseñar la registro.html.

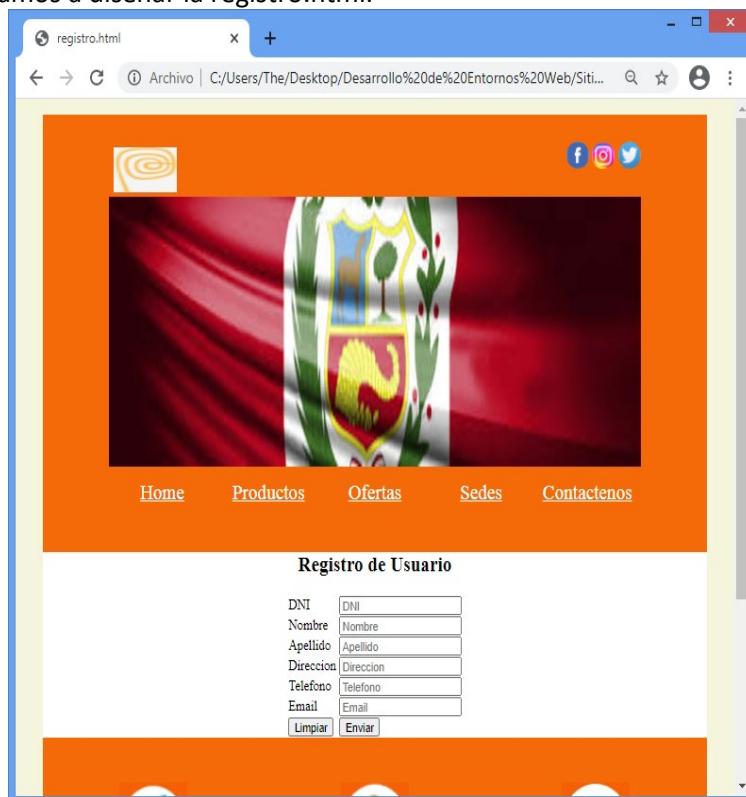


Figura 162 : Caso práctico  
Fuente.- Elaboración Propia

## Agregando los enlaces al archivo CSS y JavaScript

En la página1.html, defina el esquema de la página web. Primero enlazamos la página a la hoja de estilos: estilos.css y a los archivos JS: programa.js y validación.js. En el body, defina el esquema de la página, agregar los selectores a los elementos de esta.

```

1 <!DOCTYPE html>
2 <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title></title>
7   <link rel="stylesheet" href="css/estilos.css" /> Enlazando al archivo CSS a través de la etiqueta link
8 </head>
9 <body>
10  <main>
11    <header> ...
12    </header>
13    <section id="soferta"> ...
14    </section>
15    <footer> ...
16    </footer>
17  </main>
18 </body>
19 </html>
20 <script src="js/programa.js"></script> Enlazando al archivo JS a través de la etiqueta script
21 <script src="js/valida.js"></script>

```

Figura 163 : Caso práctico  
Fuente.- Elaboración Propia

**Diseño de la página web**

Defina los elementos de la página, tal como se muestra.

```
<body>
  <main>
    <header>
      <div id="divlogo">
        
      </div>
      <div id="divsocial">
        
        
        
      </div>
      
      <div id="nav">
        <a href="#">Home</a>
        <a href="#">Productos</a>
        <a href="#">Ofertas</a>
        <a href="#">Sedes</a>
        <a href="#">Contactenos</a>
      </div>
    </header>
    <section id="soferta">
      <h2 id="h2cab">Registro de Usuario</h2>
      <form name="frmregistro" onsubmit="registrar(this)">
        <table>
          <tr>
            <td>DNI</td>
            <td><input type="text" name="txtdni" placeholder="DNI"></td>
          </tr>
          <tr>
            <td>Nombre</td>
            <td><input type="text" name="txtnombre" placeholder="Nombre"></td>
          </tr>
          <tr>
            <td>Apellido</td>
            <td><input type="text" name="txtapellido" placeholder="Apellido"></td>
          </tr>
          <tr>
            <td>Direccion</td>
            <td><input type="text" name="txtdireccion" placeholder="Direccion"></td>
          </tr>
          <tr>
            <td>Telefono</td>
            <td><input type="text" name="txtfono" placeholder="Telefono"></td>
          </tr>
          <tr>
            <td>Email</td>
            <td><input type="text" name="txtemail" placeholder="Email"></td>
          </tr>
        </table>
      </form>
    </section>
  </main>
</body>
```

```
<td><input type="reset" value="Limpiar"></td>
<td><input type="submit" value="Enviar"></td>
</tr>
</table>
</form>
</section>
<footer>
<div class="divfooter" >
    
    <p>Tradiciones</p>
</div>
<div class="divfooter">
    
    <p>Gastronomía</p>
</div>
<div class="divfooter">
    
    <p>Festival</p>
</div>
    
</footer>
</main>
</body>
```

### Definiendo el archivo estilos.css

Definida el esquema de la página, defina el estilo a los elementos de la página en la hoja de estilo, tal como se muestra.

```
body
{
    background-color:beige;
}

main{
    width:94%;
    margin-left:3%;
    margin-right:3%;
    margin-top: 10px;
    float:left;
    background-color:white;
    border-radius:20px;
}

header{
    width:80%;
    float:left;
    background-color: #F56A09;
    padding:30px 10% 30px 10%;
}

#imgheader{
```

```
width:100%;  
height:300px;  
float:left;  
}  
  
#nav{  
    width:100%;  
    float:left;  
    margin-top:5px;  
}  
#divlogo{  
    width:40%;  
    float:left;  
}  
  
#divlogo>img{  
    width:30%;  
    height:50px;  
    float:left;  
    margin:2%  
}  
#divsocial{  
    width:100%;  
    text-align: right;  
}  
.social{  
    width: 30px;  
    width: 30px;  
    border-radius: 20px;  
}  
#nav>a{  
    height:40px;  
    padding-top: 10px;  
    padding-bottom: 10px;  
    width: 20%;  
    font-size:1.5em;  
    color: white;  
    text-align: center;  
    float: left;  
}  
#nav>a:hover{  
    background-color: white;  
    color:#F56A09;  
}  
#slogueo>h2{  
    text-align: center;  
}  
  
table{  
    margin: 0 auto;  
}  
footer{
```

```

width:100%;
min-height: 200px;
float:left;
background-color: #F56A09;
}
.divfooter{
width: 33.3%;
float: left;
text-align: center;
}
.divfooter>img{
width: 90px;
height: 90px;
padding-top: 50px;
padding-bottom: 10px;
}
.divfooter>p{
color:white;
font-size: 1.5em;
}
#imgfooter{
width: 100%;
height: 120px;
float: left;
}

```

### Definiendo la programación: programa.js.

4. En el archivo de JavaScript programa.js, defina el Array lista, el cual almacena las direcciones de las imágenes. Este Array() será utilizado en la función carrusel(), el cual cambia la imagen por intervalo de un segundo, tal como se muestra.

```

1 var lista=new Array("imgs/header1.jpg","imgs/header2.jpg",
2 | | | "imgs/header3.jpg","imgs/header4.jpg");
3
4 var c=0;
5
6 function carrusel(){
7     var img=document.getElementById("imgheader");
8     c++;
9     if(c>=lista.length) c=0;
10    img.src=lista[c];
11
12    setTimeout("carrusel()",1000);
13 }
14

```

Figura 164 : Caso práctico  
Fuente.- Elaboración Propia

5. Defina las funciones cambio(e) y regresa(e), funciones que permiten cambiar la imagen de la red social. A continuación, defina las funciones gira(e) y retorna(e) funciones que permiten girar un elemento 360 grados y 0 grados en su eje Y por un segundo.

```

14
15  function cambio(e){
16  |  var img=e.getAttribute("data-social");
17  |  e.src="imgs/"+img+"1"+".jpg";
18 }
19
20 function regresa(e){
21 |  var img=e.getAttribute("data-social");
22 |  e.src="imgs/"+img+".jpg";
23 }
24
25 function gira(e){
26 |  e.style.transform="rotateY(360deg)";
27 |  e.style.transition="all 1s";
28 }
29
30 function retorna(e){
31 |  e.style.transform="rotateY(0deg)";
32 |  e.style.transition="all 1s";
33 }
34

```

Ln 37, Col 1 Tab Size:4 UTF-8 CRLF JavaScript ⚙️ ⚙️

Figura 165 : Caso práctico

Fuente.- Elaboración Propia

6. A continuación, en el archivo programa.js, enlazamos a los elementos de las páginas, los eventos correspondientes, tal como se muestra.

```

29
30 > function retorna(e){ ...
31 }
32
33 /*eventos*/
34 document.body.setAttribute("onload", "carrusel()");
35
36 for(var i=0; i<document.getElementsByClassName("social").length; i++){
37 {
38   document.getElementsByClassName("social")[i].setAttribute("onmouseover", "cambio(this)");
39   document.getElementsByClassName("social")[i].setAttribute("onmouseout", "regresa(this)");
40 }
41
42 for(var i=0; i<document.getElementsByClassName("imgfooter").length; i++){
43   document.getElementsByClassName("imgfooter")[i].setAttribute("onmouseover", "gira(this)");
44   document.getElementsByClassName("imgfooter")[i].setAttribute("onmouseout", "retorna(this)");
45 }
46
47 /*fin de eventos*/
48
49

```

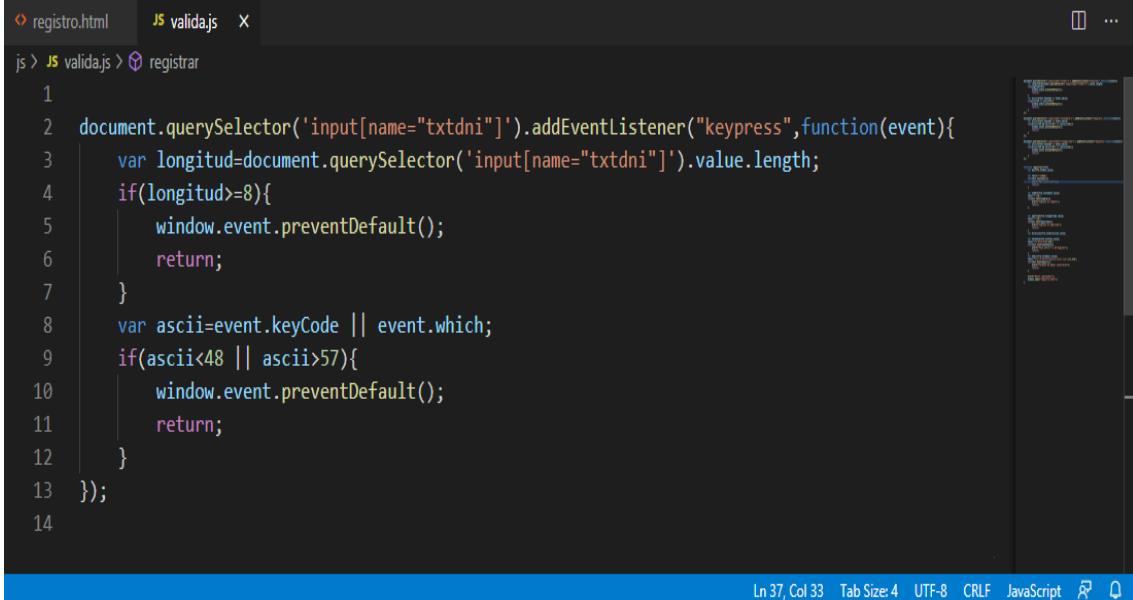
Ln 14, Col 1 Tab Size:4 UTF-8 CRLF JavaScript ⚙️ ⚙️

Figura 166 : Caso práctico

Fuente.- Elaboración Propia

### Definiendo la programación: valida.js.

1. Defina el evento keypress para el input txtdni, donde verifique que la longitud de este sea 8 caracteres; y además solo permite el ingreso de caracteres numéricos.



```

registro.html JS valida.js X
js > JS valida.js > registrar
1
2 document.querySelector('input[name="txtdni"]').addEventListener("keypress",function(event){
3     var longitud=document.querySelector('input[name="txtdni"]').value.length;
4     if(longitud>=8){
5         window.event.preventDefault();
6         return;
7     }
8     var ascii=event.keyCode || event.which;
9     if(ascii<48 || ascii>57){
10        window.event.preventDefault();
11        return;
12    }
13 });
14

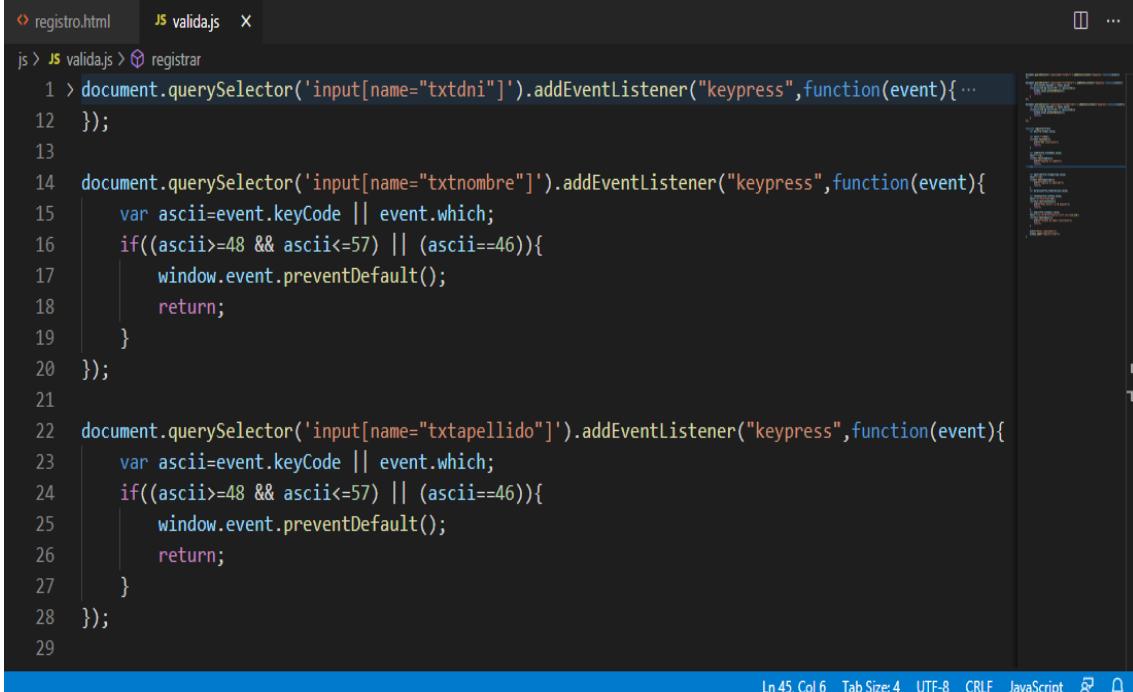
```

Ln 37, Col 33 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ ⚙️

Figura 167: Caso práctico

Fuente.- Elaboración Propia

2. Defina el evento keypress para los inputs txtnombre y txtapellido, donde permite solamente el ingreso de caracteres no numéricos.



```

registro.html JS valida.js X
js > JS valida.js > registrar
1 > document.querySelector('input[name="txtdni"]').addEventListener("keypress",function(event){ ...
12 });
13
14 document.querySelector('input[name="txtnombre"]').addEventListener("keypress",function(event){
15     var ascii=event.keyCode || event.which;
16     if((ascii>=48 && ascii<=57) || (ascii==46)){
17         window.event.preventDefault();
18         return;
19     }
20 });
21
22 document.querySelector('input[name="txtapellido"]').addEventListener("keypress",function(event){
23     var ascii=event.keyCode || event.which;
24     if((ascii>=48 && ascii<=57) || (ascii==46)){
25         window.event.preventDefault();
26         return;
27     }
28 });
29

```

Ln 45, Col 6 Tab Size: 4 UTF-8 CRLF JavaScript ⚙️ ⚙️

Figura 168 : Caso práctico

Fuente.- Elaboración Propia

3. Defina el método registrar, donde valida los datos del input utilizando las expresiones regulares, dichas validaciones están especificadas en la pregunta.

```
registro.html  JS valida.js ...  
js > JS valida.js > ...  
31 function registrar(frm){  
32     let test=/^\d{8}$/;  
33     if(!test.test(frm.txtdni.value)){  
34         window.event.preventDefault();  
35         alert("DNI incorrecto"); return;  
36     }  
37     test=/^\s*$/;  
38     if(test.test(frm.txtnombre.value)){  
39         window.event.preventDefault();  
40         alert("Ingrese el nombre"); return;  
41     }  
42     if(test.test(frm.txtapellido.value)){  
43         window.event.preventDefault();  
44         alert("Ingrese el apellido"); return;  
45     }  
46     test=/^([2-9][0-9]{7,9})$/;  
47     if(!test.test(frm.txtfono.value)){  
48         window.event.preventDefault();  
49         alert("Fono entre 7 a 10 digitos"); return;  
50     }  
51     test=/^([a-zA-Z0-9])+@[.][a-zA-Z]{2,3}$/;  
52     if(!test.test(frm.txtemail.value)){  
53         window.event.preventDefault();  
54         alert("formato de email incorrecto"); return;  
55     }  
56     alert("Datos ingresados");  
57 }  
58
```

Ln 21, Col 1 Tab Size 4 UTF-8 CRLF JavaScript ⚡ ⚡

Figura169 : Caso práctico  
Fuente.- Elaboración Propia

Para ejecutar la página web, seleccionar la pestaña registro.html, y presiona la tecla F5.

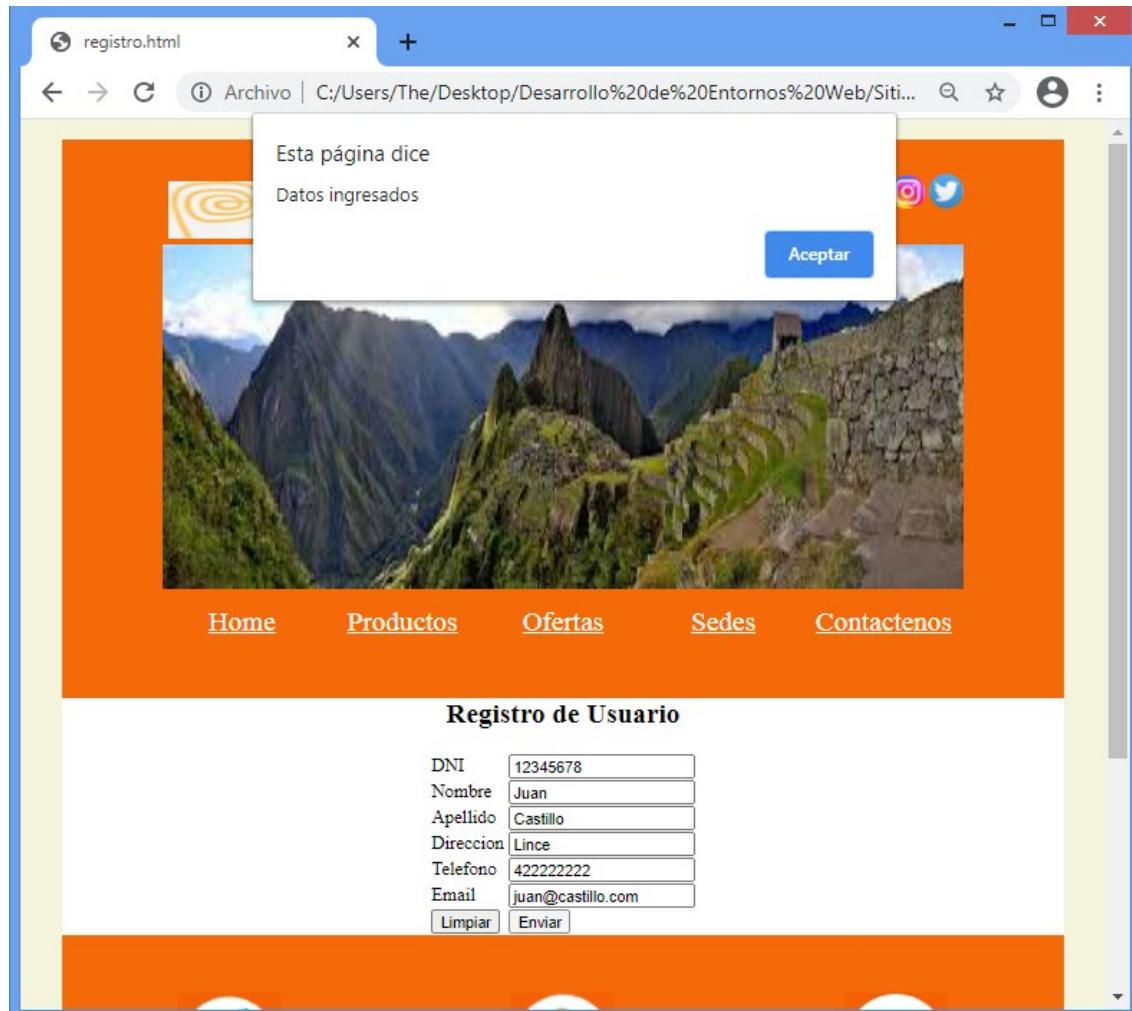


Figura 170 : Caso práctico  
Fuente.- Elaboración Propia

# Resumen

1. Las expresiones regulares son secuencia de caracteres que forman un patrón para encontrar o reemplazar una determinada combinación de caracteres dentro de una cadena de texto.
2. En JavaScript, las expresiones regulares también son objetos. Estos patrones son utilizados a través de los métodos exec y test del objeto RegExp.
3. Las expresiones regulares son muy útiles para validar formularios.

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [https://www.youtube.com/watch?v=93Ha7qd\\_qvo](https://www.youtube.com/watch?v=93Ha7qd_qvo)
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular\\_Expressions](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions)
- [http://www.w3schools.com/jsref/jsref\\_obj\\_regexp.asp](http://www.w3schools.com/jsref/jsref_obj_regexp.asp)
- [http://chuwiki.chuidiang.org/index.php?title=Expresiones\\_regulares\\_en\\_JavaScript](http://chuwiki.chuidiang.org/index.php?title=Expresiones_regulares_en_JavaScript)



# REACT JS

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno, utilizando el framework React JS, desarrollo aplicaciones web con SPA (Single Page Application) y aplicar a los proyectos existentes.

## TEMARIO

### 6.1 Tema 9 : Introducción

- 6.1.1 : Definición
- 6.1.1.1 : Componentes
- 6.1.1.2 : Virtual DOM
- 6.1.2 : Configurando e instalando el entorno react
- 6.1.3 : Nuestra primera aplicación con react desde un entorno vacío

### 6.2 Tema 10 : Trabajando con react

- 6.2.1 : componentes react con createclass
- 6.2.2 : Componentes react mediante clases ES6
- 6.2.3 : Propiedades y estados en componentes react

### 6.3 Tema 11 : Ciclo de vida y eventos en react

- 6.3.1 : Ciclo de vida de los componentes
- 6.3.2 : Eventos en react
- 6.3.3 : Condicionales en templates JSX de react
- 6.3.4 : Creación de repeticiones en templates JSX con react
- 6.3.5 : Extendiendo componentes con mixins

## ACTIVIDADES PROPUESTAS

- Los alumnos aprende a configurar React.
- Los alumnos define componentes para diseñar una página html.
- Los alumnos renderiza la aplicación para su ejecución.

## 6.1. INTRODUCCIÓN

Al momento de escoger cuál tecnología usar en el frontend de un proyecto nuevo, nos enfrentamos a una delicada e importante decisión que va a influir mucho en el futuro de nuestra aplicación, por lo que es importante escoger tecnologías que complementen y faciliten el desarrollo.

### 6.1.1 Definición

La web está construida con 3 lenguajes básicos:

- HTML es la estructura e información de la página web y es completamente estático.
- CSS define la apariencia (estilos) a los elementos de la página HTML, y se adapte a todos los tamaños de pantalla (responsive).
- JavaScript es el lenguaje de programación.

React propuso colocar a todos en un solo paquete llamado componente, donde HTML y JavaScript siempre están unidos, opcionalmente puedes escribir CSS dentro del componente o puedes trabajar de manera tradicional (Por fuera).

React es una biblioteca escrita en JavaScript, desarrollada en Facebook para facilitar la creación de componentes interactivos, reutilizables, para interfaces de usuario. Se utiliza en Facebook para la producción de componentes, e Instagram está escrito enteramente en React. Uno de sus puntos más destacados, es que no sólo se utiliza en el lado del cliente, sino que también se puede representar en el servidor, y trabajar juntos.

React aporta una serie de ventajas frente a la forma clásica de realizar una web, sus facilidades para el desarrollo unido al rendimiento, la flexibilidad y organización del código la convierten en una de las mejores opciones:

- a. React está basado en un paradigma de programación orientada a componentes en el que cada componente es una pieza con la que el usuario puede interactuar. Estas piezas se crean usando una sintaxis llamada JSX permitiendo escribir HTML (y opcionalmente CSS) dentro de objetos JavaScript.
- b. Sus componentes son reutilizables y se combinan para crear componentes mayores hasta configurar una web completa.
- c. Permite la encapsulación del código en componentes, que nos ofrecen una serie de ventajas más importantes que los plugin, como la posibilidad de que esos componentes conversen e interactúen entre sí, algo que sería muy difícil de conseguir con Plugins.
- d. Es capaz de generar el DOM de forma dinámica (virtual DOM), hace los cambios en una copia en memoria y después la compara con la versión actual del DOM, de esta forma evita renderizar toda la página cada vez que haya cambios, simplemente se aplica dicho cambio al componente que haya sido actualizado, simple y rápido. Esto propicia una mejor experiencia de usuario, además de un rendimiento y una fluidez impresionante.
- e. Con respecto a librerías, React aporta una serie de posibilidades muy importantes. Al tener las vistas asociadas a los datos, no necesitamos escribir código para manipular la página cuando los datos cambian. Esta parte en librerías sencillas es muy laboriosa de conseguir y es algo que React hace automáticamente.

### 6.1.1.1 Componentes

Un componente en React es un elemento independiente que tiene las siguientes características:

- Son piezas de UI (User Interface). Cada pieza en la pantalla con la que el usuario interactúa es un componente.
- Son reutilizables y combinarse para crear componentes mayores que a su vez se combinen crean un sistema completo.
- Son objetos, la lógica y la estructura están contenidos en un objeto de JavaScript
- Los componentes se crean con JSX. Este el secreto de React.js, nos permite escribir HTML con la misma sintaxis dentro de JavaScript. Webpack se encarga de convertir este código a JavaScript estándar que el navegador puede interpretar.

Existen dos tipos de componentes en React:

Componentes funcionales: Son componentes que generan elementos React. Por convenio se pone el nombre de la función en mayúsculas. Para renderizarlo simplemente se pone una etiqueta con el nombre de la función.

Componentes de clase: Tienen propiedades, ciclos de vida y propiedades.

### 6.1.1.2 Virtual DOM

El DOM es una estructura HTML de un sitio.

Cuando cargas una página web, el navegador interpreta el código HTML, CSS y JavaScript; y lo pinta en pantalla. Cuando hay un cambio en el DOM el navegador tiene que repetir todo el proceso y es muy costoso en términos de recursos.

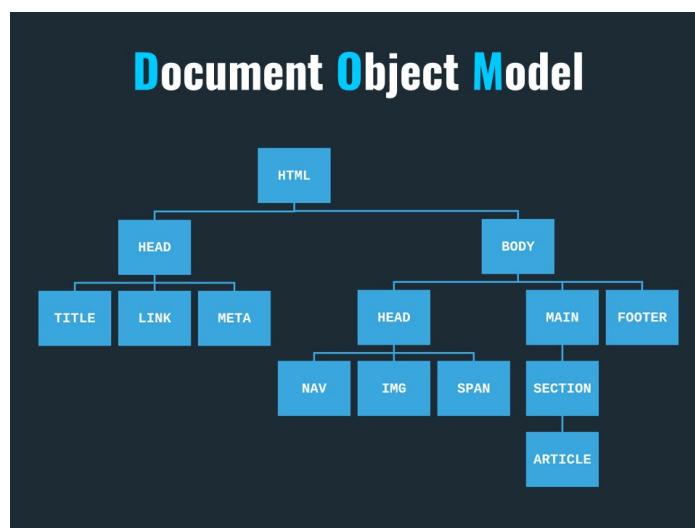


Figura 171: Virtual DOM

Fuente. - Tomado de <https://ed.team/blog/como-funciona-reactjs>

El Virtual DOM se basa en una idea bastante sencilla e ingeniosa. Básicamente hace que, cuando se actualiza una vista, React se encargue de actualizar el DOM Virtual, que es mucho más rápido que actualizar el DOM del navegador (DOM real).

Cuando React compara el DOM Virtual con el DOM del navegador sabe perfectamente qué partes de la página debe actualizar y se ahorra la necesidad de actualizar la vista entera. Es algo muy potente, pero que se hace de manera transparente para el desarrollador, que no necesita intervenir en nada para alcanzar ese mayor rendimiento de la aplicación.

### 6.1.2. Configurando e instalando el entorno React

Para instalar el entorno React, empecemos instalando la versión recomendada de NodeJS que puedes encontrar y descargar desde su sitio oficial: <https://nodejs.org>.

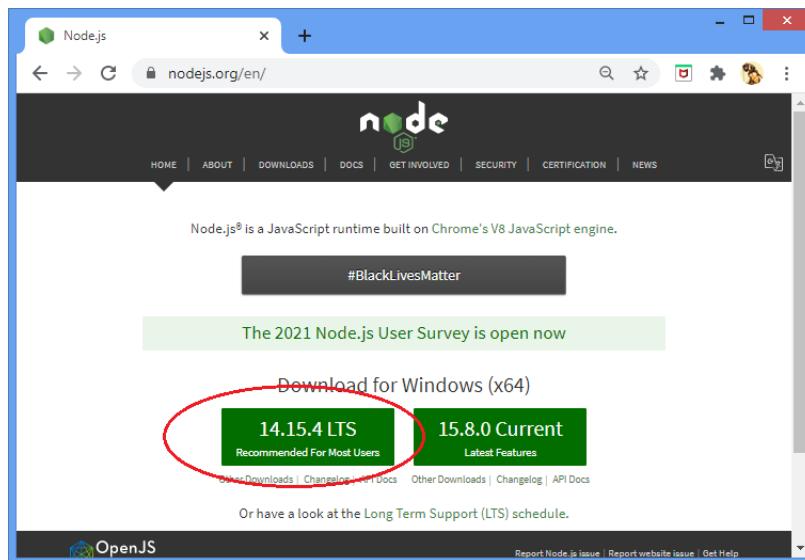


Figura 172: Configurando e instalando el entorno React

Fuente.- Tomado de <https://nodejs.org/en>

Al finalizar la instalación, abrimos nuestra terminal y comprobamos que la instalación fue exitosa escribiendo en la consola node --version, lo que nos arrojará la versión activa de NodeJS.

```
C:\Users\The\Desktop\sesion01react>node --version
v14.15.4
C:\Users\The\Desktop\sesion01react>
```

Figura 173: Ventana de comando

Fuente.- Elaboración Propia

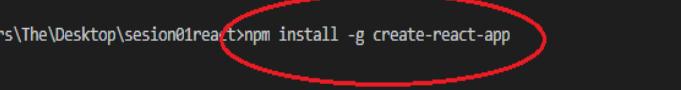
Al instalar NodeJS, también estamos instalando NPM, que es el manejador de paquetes de Node. Esto nos va a ser de utilidad para instalar dependencias en nuestros proyectos. Para comprobar que se ha instalado correctamente, intenta escribir el comando npm -version en la consola para ver la versión activa de NPM.

C:\Users\The\Desktop\sesion01react>npm --version  
6.14.10

C:\Users\The\Desktop\sesion01react>

Figura 174: Ventana de comando  
Fuente.- Elaboración Propia

Una vez instalado exitosamente el NodeJS, procedemos a instalar **create-react-app**.



```
C:\Users\The\Desktop\sesion01react>npm install -g create-react-app
```

Figura 175: Ventana de comando  
Fuente.- Elaboración Propia

Es una herramienta desarrollada por Facebook para comenzar a crear proyectos en React de una forma sencilla. Con esta herramienta, podrás comenzar a escribir código React en un par de minutos, ya que es como una especie de Boilerplate que tiene todo para que comiences a codear. Otra ventaja de `create-react-app` es que ya tiene configurado webpack y babel, además de que funciona con los sistemas operativos más populares.

Para instalarlo solo basta con correr el comando `npm install -g create-react-app` en Windows.

Figura 176: Ventana de comando

### 6.1.3 Nuestra primera aplicación con React, desde un entorno vacío

Para crear una app con `create-react-app`, basta con abrir la terminal y navegar hasta el directorio deseado. Una vez ahí, corremos el comando `create-react-app` seguido del nombre de la app.

The screenshot shows a terminal window with the following commands:

```
C:\Users\The>cd Desktop
C:\Users\The\Desktop>create-react-app myapp
```

Annotations explain the steps:

- A red box highlights "C:\Users\The>cd Desktop". A callout bubble says "seleccionar el directorio Desktop".
- A red box highlights "create-react-app myapp". A callout bubble says "creando el app llamado myapp".

Figura 177: Ventana de comando

Fuente.- Elaboración Propia

Va a crear un directorio por nosotros en donde vivirá nuestra app e instalará todas las dependencias necesarias para empezar a crear la aplicación.

The screenshot shows a terminal window with the following output:

```
npm run build
  Bundles the app into static files for production.

npm test
  Starts the test runner.

npm run eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd myapp
  npm start

Happy hacking!
```

The prompt at the bottom is:

```
C:\Users\The\Desktop>
```

Figura 178: Ventana de comando

Fuente.- Elaboración Propia

## LABORATORIO 1

### Instalando React JS

En este laboratorio vamos a aprender a instalar React JS en nuestro computador.

#### Node.js

1. Instalamos, primero, Node.js, por ser un intérprete de JavaScript, es decir, permite ejecutar JavaScript en tu sistema operativo. Es necesario en todos los framework de JavaScript modernos.

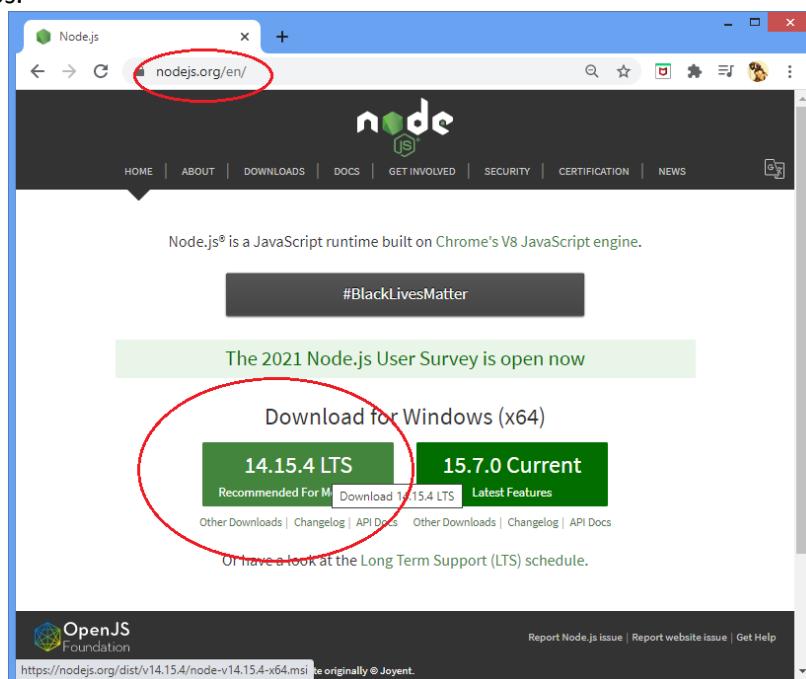


Figura 179: NodeJS  
Fuente.- Elaboración Propia

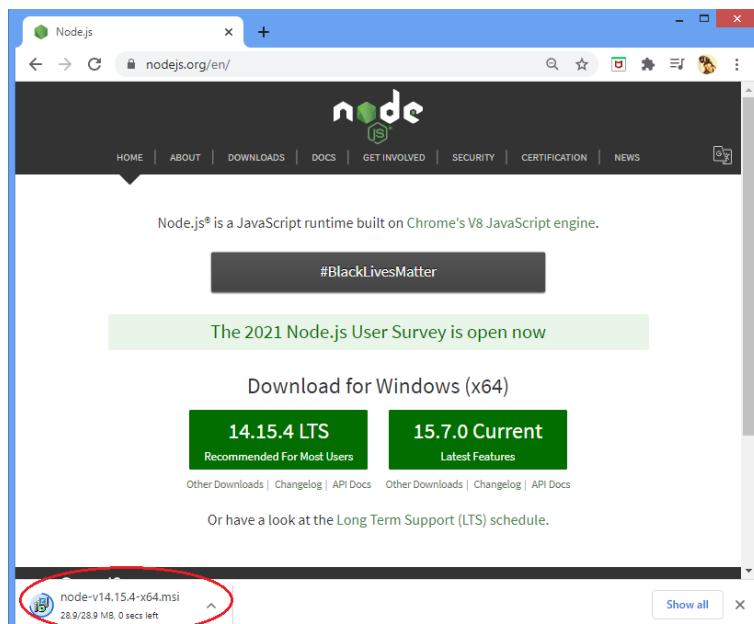


Figura 180: NodeJS  
Fuente.- Elaboración Propia

2. Seguir los pasos para la instalación, presionar Next. En la siguiente ventana **Aceptar los términos** y presionar Next.

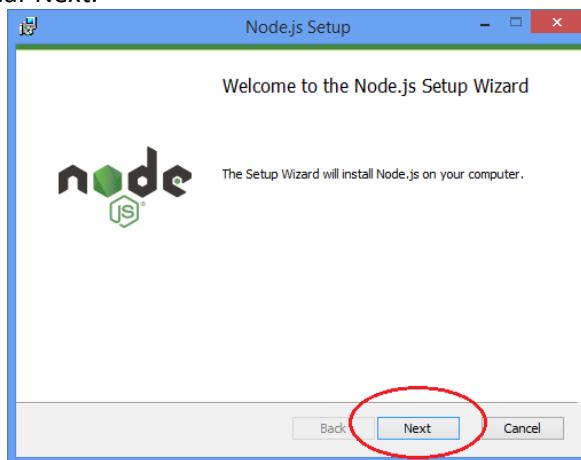


Figura 181: NodeJS  
Fuente.- Elaboración Propia

3. Seleccionar el destino de los archivos, conservar la ubicación y presionar Next

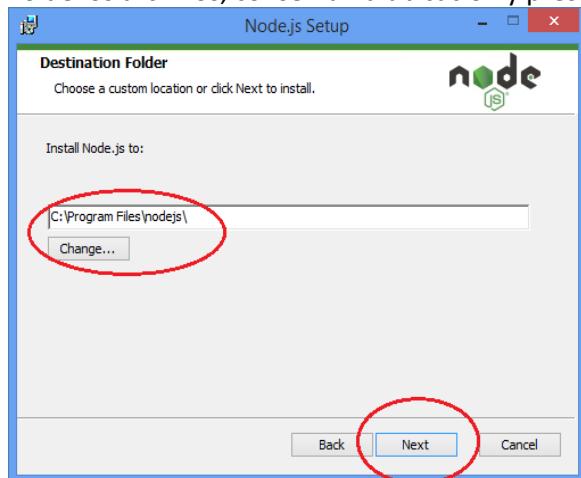


Figura 182: NodeJS  
Fuente.- Elaboración Propia

4. Seleccionar las opciones de instalación, conserver todas y presionar Next

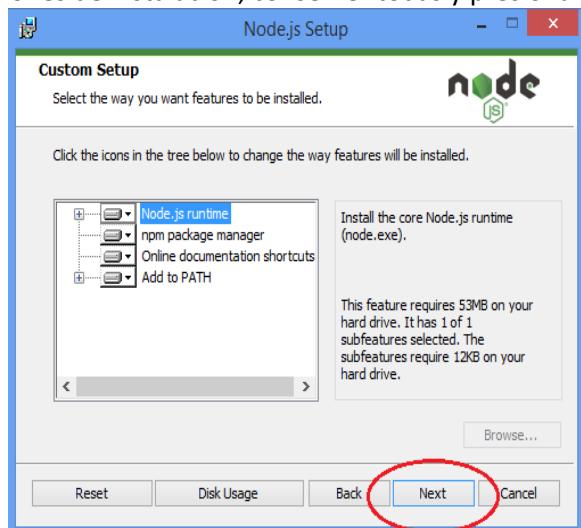


Figura 183 : NodeJS  
Fuente.- Elaboración Propia

5. Al finalizar el asistente, presionar el botón Install para iniciar la instalación, donde instalarás los paquetes. Al finalizar presionar el botón Finish

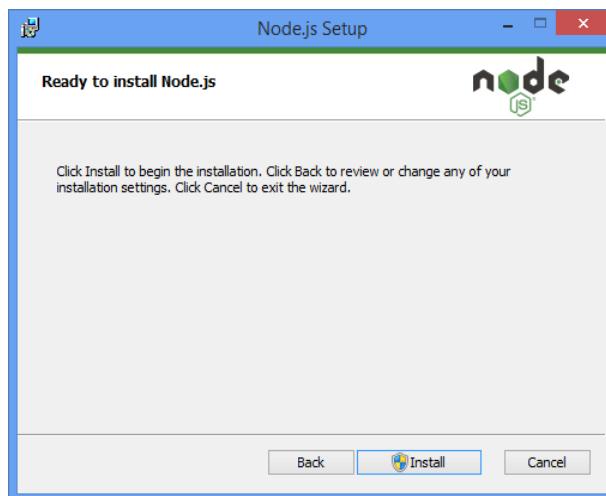


Figura 184: NodeJS  
Fuente.- Elaboración Propia

## React.js

1. Para instalar React, vamos a la página <https://reactjs.org>, desde el link seleccionamos el idioma a español.  
A continuación, selecciona la opción Documentación y seleccionamos, desde instalación, la opción Crear una nueva aplicación React, tal como se muestra.

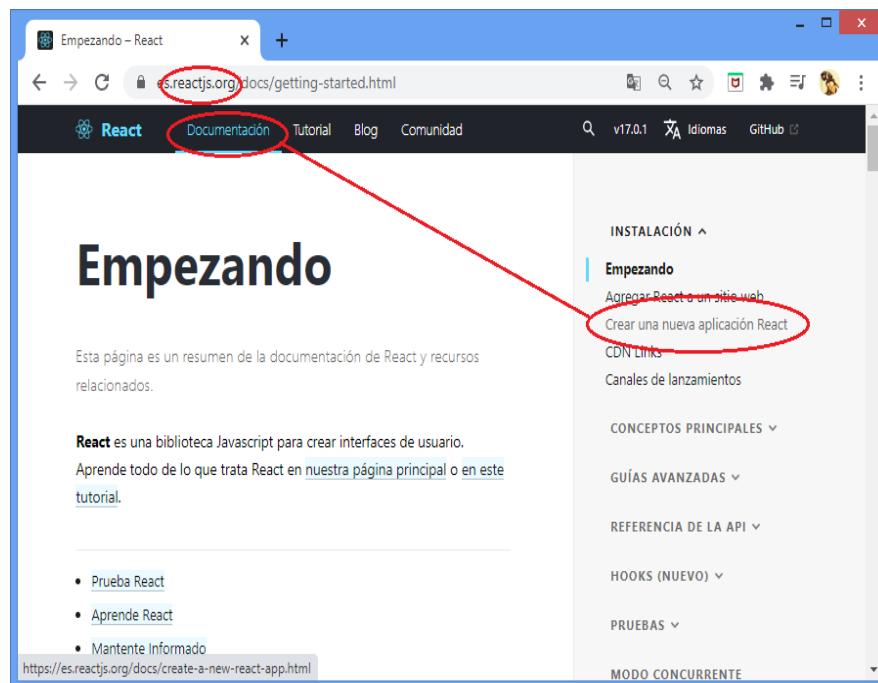


Figura 185: React JS  
Fuente.- Elaboración Propia

2. Selecciona la opción, ir a Cadenas de herramientas recomendadas y hacer click a la opción: **Create React App**, tal como se muestra

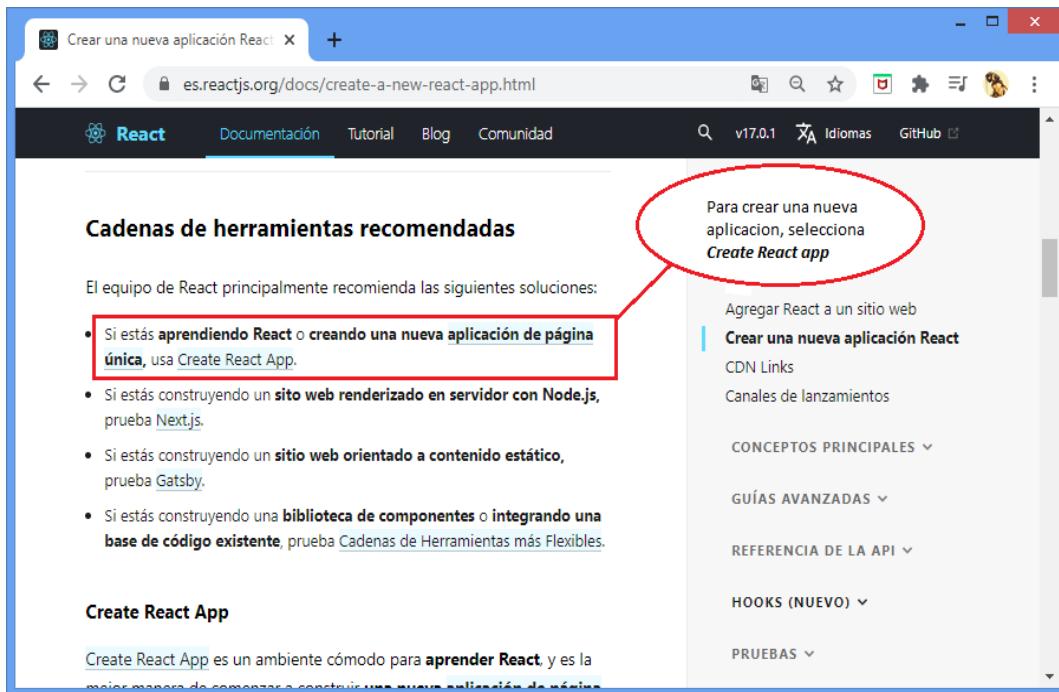


Figura 186: React JS  
Fuente.- Elaboración Propia

3. Al tener instalado el Node.js (el cual ejecuta el comando npm), a continuación, creamos nuestra aplicación React

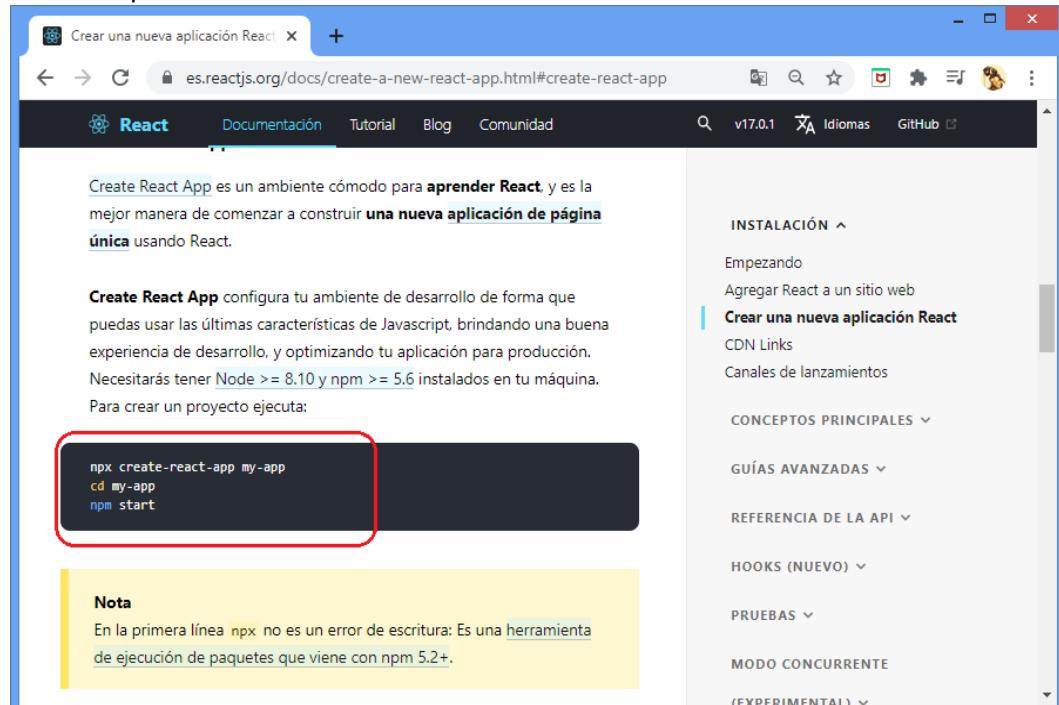


Figura 187: React JS  
Fuente.- Elaboración Propia

4. Abrir el Visual Studio Code, y presiona la combinación Ctrl + Shift + P, allí escribe Terminal: **Create New Integrated Terminal** y seleccionar para visualizar una ventana de comando (recuerda que debe ser cmd)

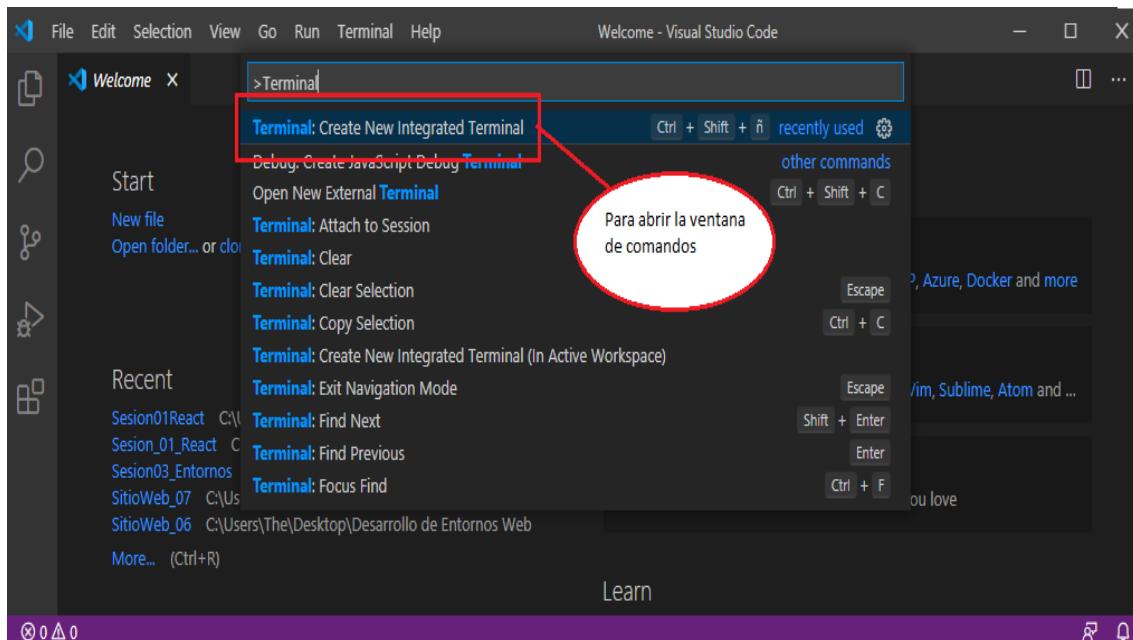


Figura 188: Visual Studio Code  
Fuente.- Elaboración Propia

5. Para saber la versión del node instalado escribir node --version y presionar ENTER.

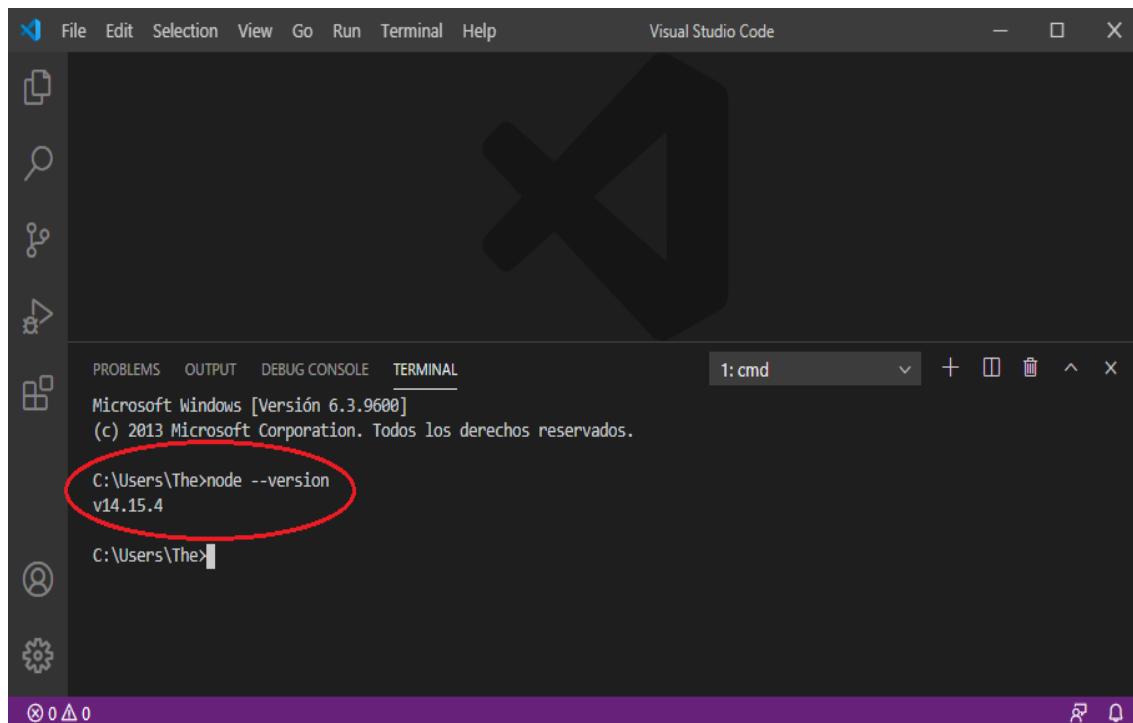
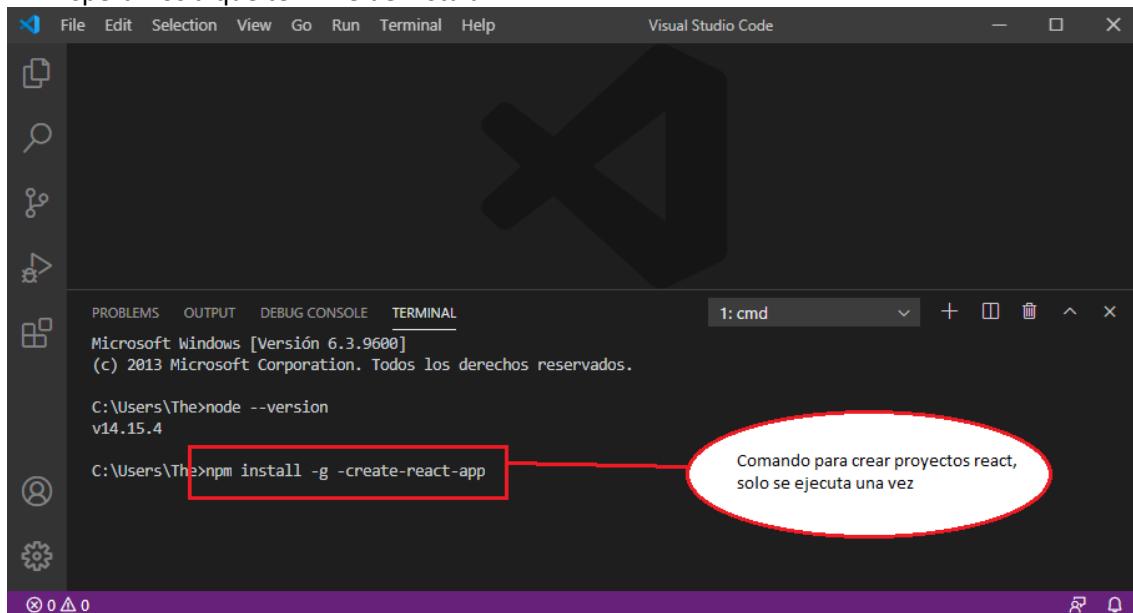


Figura 189: Visual Studio Code  
Fuente.- Elaboración Propia

6. A continuación, vamos a descargar un nuevo comando para el Sistema Operativo, este comando nos permite generar proyecto React: **npm install -g create-react-app**  
Esperamos a que termine de instalar...



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal tab is selected and contains the following text:

```
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

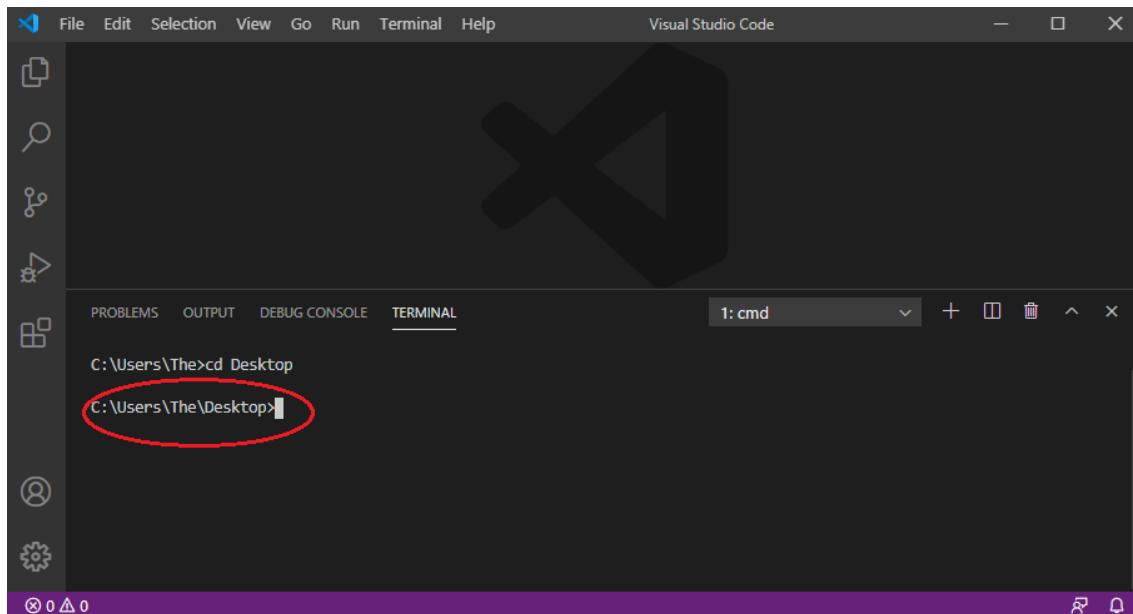
C:\Users\The>node --version
v14.15.4

C:\Users\The>npm install -g -create-react-app
```

A red box highlights the command "npm install -g -create-react-app". A red oval with the text "Comando para crear proyectos react, solo se ejecuta una vez" is positioned over the terminal window, pointing to the highlighted command.

Figura 190: Visual Studio Code  
Fuente.- Elaboración Propia

7. Para crear un nuevo proyecto en el Escritorio, Utilice el comando cd, ubicarse en el Escritorio



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal tab is selected and contains the following text:

```
C:\Users\The>cd Desktop
```

A red oval highlights the command "cd Desktop".

Figura 191: Visual Studio Code  
Fuente.- Elaboración Propia

8. Ejecutando el comando `create-react-app` creamos el proyecto `sesion01react` (escribir en minúsculas) e instalamos las herramientas del proyecto react.

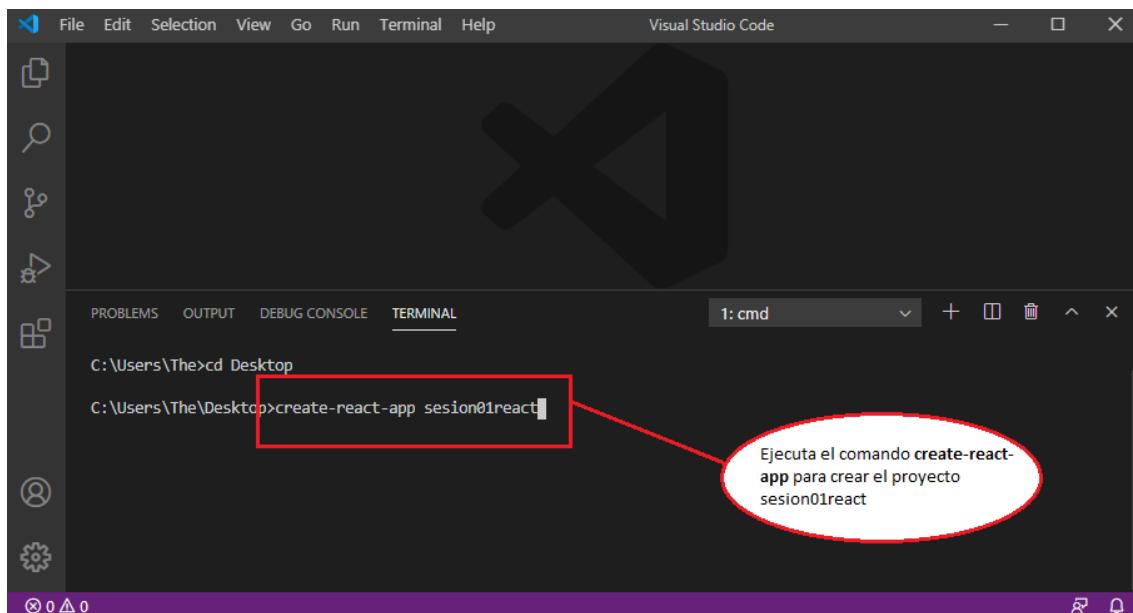


Figura 192: Visual Studio Code  
Fuente.- Elaboración Propia

9. Creada la aplicación, presionar el botón Open Folder, seleccionar la carpeta del proyecto y presionar el botón Seleccionar Carpeta.

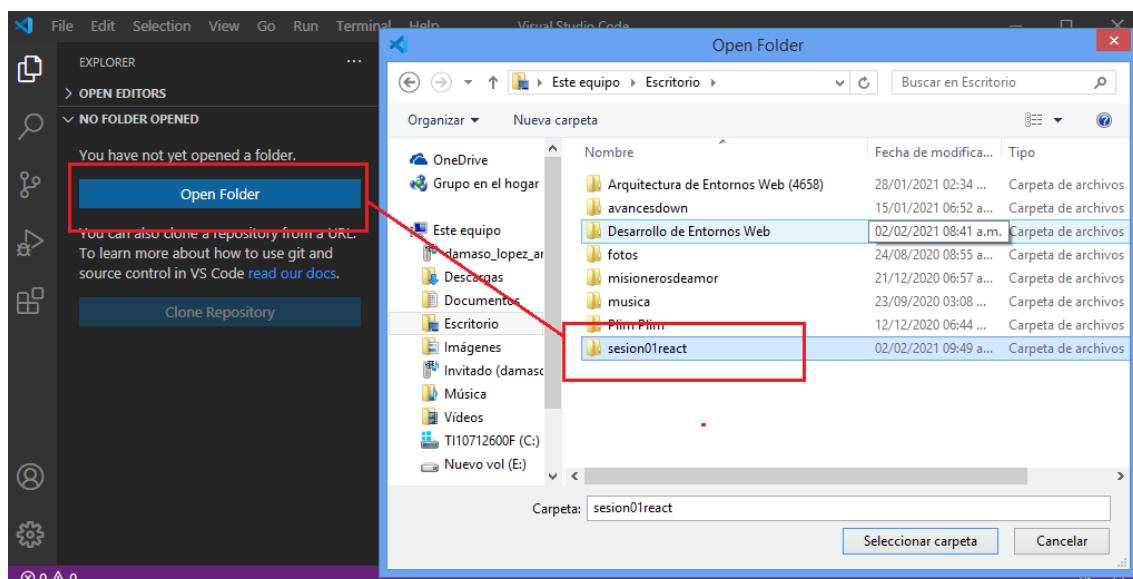


Figura 193: Visual Studio Code  
Fuente.- Elaboración Propia

10. Visualizamos la carpeta y sus archivos.

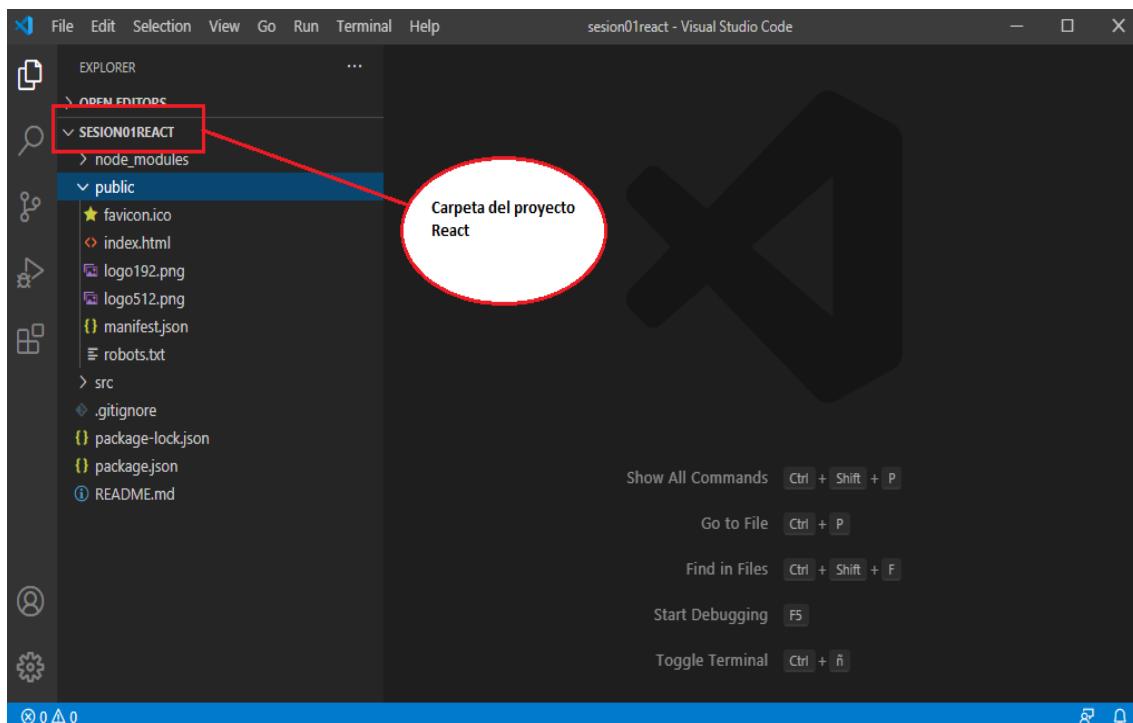


Figura 194: Visual Studio Code  
Fuente.- Elaboración Propia

## LABORATORIO 2

### Trabajando con nuestra primera aplicación en React JS

En este laboratorio vamos a aprender a instalar React JS en nuestro computador.

1. Creado el proyecto en el Laboratorio01, vamos a ejecutar el proyecto, a través de la ventana de comando, escriba npm start y presionar ENTER

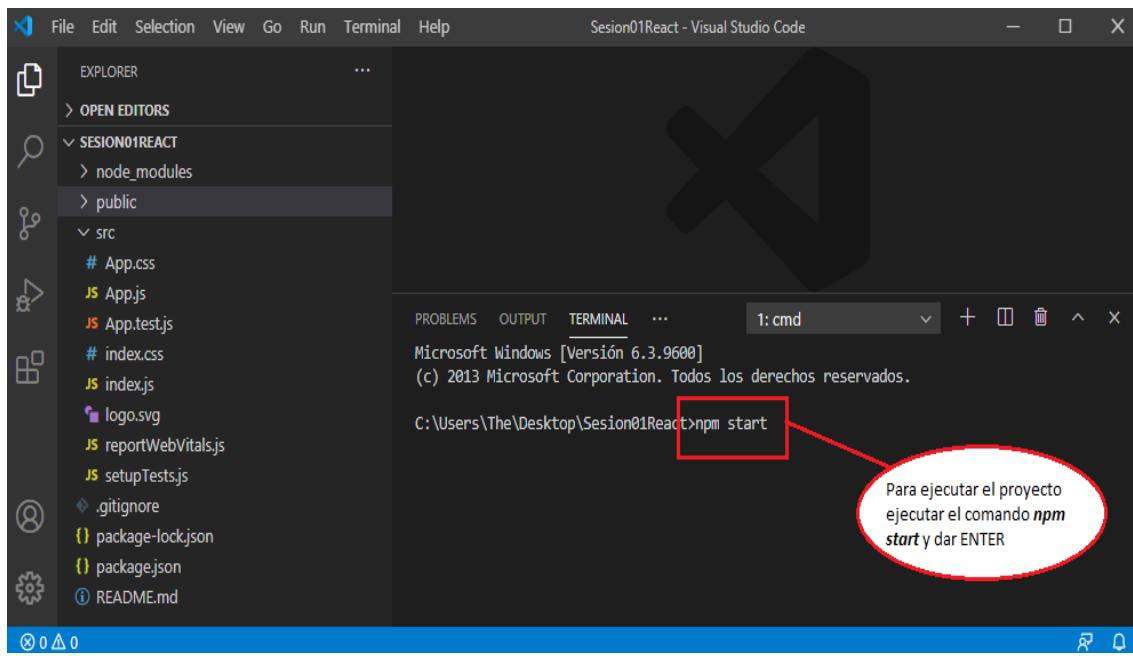


Figura 195: Proyecto de Visual Studio Code

Fuente.- Elaboración Propia

2. Ejecuta un servidor de aplicaciones en el localhost:3000

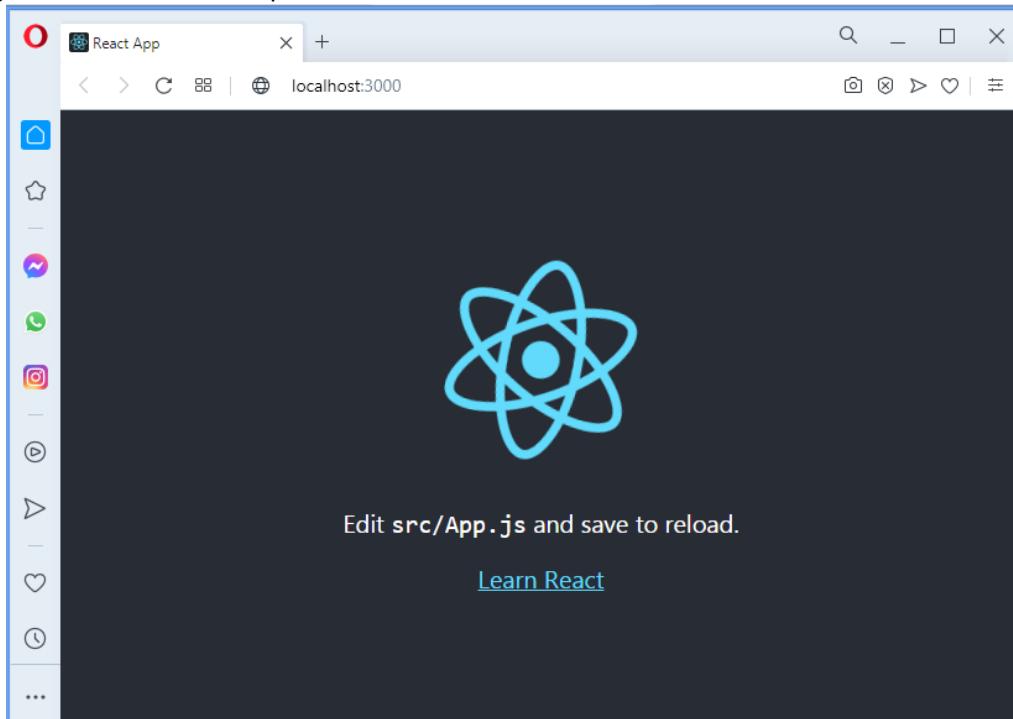


Figura 196: Proyecto de Visual Studio Code

Fuente.- Elaboración Propia

## Implementando nuestra primera aplicación.

Utilizando componentes en React, diseña la página index.html, tal como se muestra en la figura.

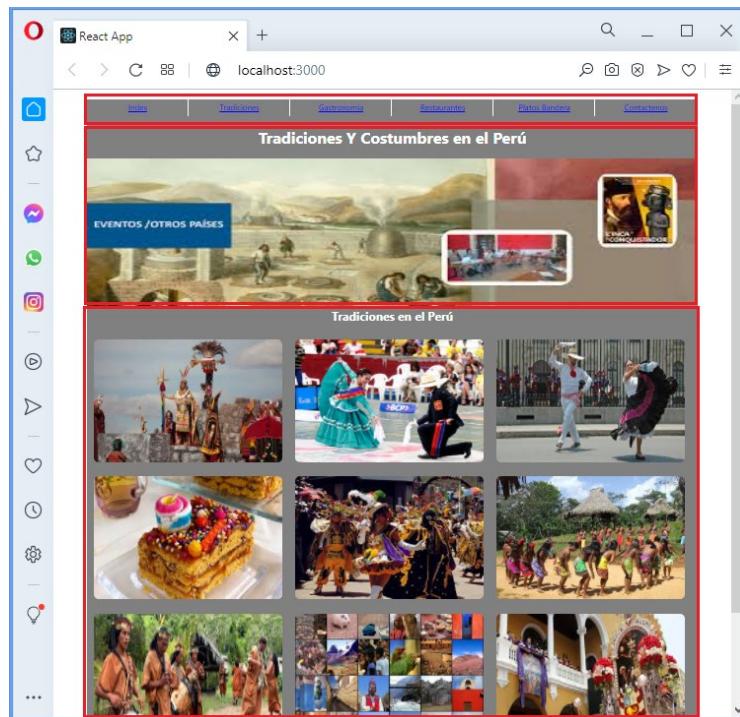


Figura 197 : Caso práctico  
Fuente.- Elaboración Propia

## Trabajando con la página index.js

Esta página es el índice del proyecto, archivo que ayuda a arrancar la aplicación.

ReactDOM.render, renderiza la aplicación y en donde lo pintamos (root) ubicado en index.html

```

EXPLORER      ...
OPEN EDITORS
SESION01REACT
> node_modules
> public
src
> img
# App.css
JS App.js
JS Header.js
JS imagenes.js
# index.css
JS index.js
JS menus.js
JS Pie.js
JS Seccion.js
JS Tradiciones.js
.eslintcache
.gitignore
package-lock.json
package.json
README.md

JS index.js  x
src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';

5
6 ReactDOM.render(
7   <React.StrictMode>
8     | <App />
9   </React.StrictMode>,
10  document.getElementById('root')
11 );
12
13

```

Ln 1, Col 27 Spaces: 2 UTF-8 LF JavaScript ⚙️ 🔍

Figura 198: Caso práctico  
Fuente.- Elaboración Propia

### Trabajando con App.js

En este archivo importamos los archivos App.css (hojas de estilo) y archivo JS (Menu, Header, Seccion y Pie). En el App estamos integrando css y JS. En la función App(), estamos invocando a cada uno de los archivos definidos.

```

EXPLORER      ...
OPEN EDITORS
SESION01REACT
node_modules
public
src
img
# App.css
JS App.js
JS Header.js
JS imagenes.js
index.css
JS index.js
JS Menu.js
JS Pie.js
JS Seccion.js
JS Tradiciones.js
.eslintcache
.gitignore
package-lock.json
package.json
README.md

JS App.js
import React from 'react';
import './App.css';
import Menu from './Menu';
import Header from './Header';
import Seccion from './Seccion';
import Pie from './Pie';

//componente creado por React
function App() {
  return (
    <div className="container">
      <Menu/>
      <Header/>
      <Seccion/>
      <Pie/>
    </div>
  );
}

export default App;

```

Figura 199 : Caso práctico  
Fuente.- Elaboración Propia

### Trabajando con App.css

En este archivo vamos a colocar los estilos que vamos a utilizar en el proyecto

```

EXPLORER      ...
OPEN EDITORS 1 UNSAVED
SESION01REACT
node_modules
public
src
img
# App.css
JS App.js
JS Header.js
JS imagenes.js
index.css
JS index.js
JS Menu.js
JS Pie.js
JS Seccion.js
JS Tradiciones.js
.eslintcache
.gitignore
package-lock.json
package.json
README.md

# App.css
# App.css > .imgseccion
> .h1cab, .h2cab{ ...
> .imgcab{ ...
> .container{ ...
16 }
> .bloque{ ...
22 }
> .item{ ...
30 }
> .item:hover{ ...
34 }
> .divseccion{ ...
39 }
> .imgseccion[ ...
45 ]
> .imgpie{ ...
49 }


```

Figura 200: Caso práctico  
Fuente.- Elaboración Propia

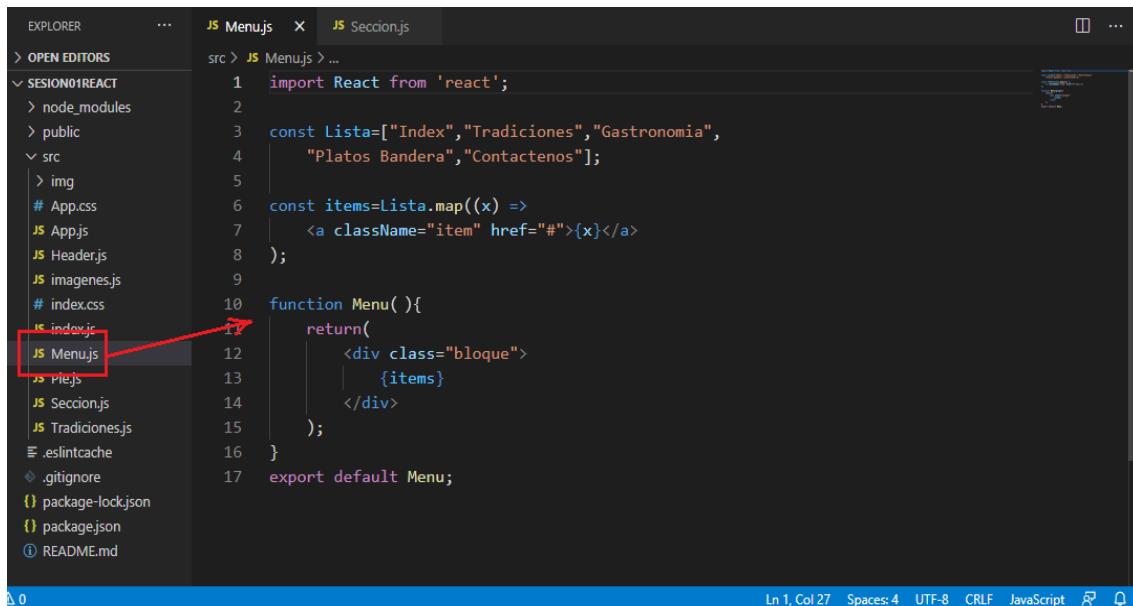
## Contenido del archivo App.css

```
.h1cab,.h2cab{  
    text-align: center;  
    color:white;  
}  
.imgcab{  
    width: 100%;  
    height: 300px;  
}  
.container{  
    width: 90%;  
    height: auto;  
    margin-top: 10px;  
    margin-bottom: 10px;  
    margin-left: 5%;  
    margin-right: 5%;  
}  
.bloque{  
    width: 100%;  
    height: auto;  
    float: left;  
    background-color: gray;  
}  
.item{  
    display: inline-block;  
    width: 16.5%;  
    text-align: center;  
    border-right: 1px solid white;  
    height: 30px;  
    padding-top: 5px;  
}  
.item:hover{  
    background-color: white;  
    color:gray;  
}  
.divseccion{  
    width: 100%;  
    height: auto;  
    background-color: gray;  
}  
.imgseccion{  
    width: 31%;  
    height: 250px;  
    margin: 1%;  
    border-radius: 10px;  
}  
.imgpie{  
    width: 100%;  
    height: auto;  
}
```

### Trabajando con el archivo Menu.js

El archivo Menu.js retorna los hipervínculos de la página, para ello definimos una lista de elementos del menú, luego defina ítems como un array de hipervínculos <a> asignando a cada título el valor de Lista y su nombre de class="ítem".

El componente Menu, envía, dentro del área llamado “bloque” los ítems.



```

EXPLORER      ...
JS Menu.js X  JS Seccion.js
OPEN EDITORS
src > JS Menu.js > ...
1 import React from 'react';
2
3 const Lista=["Index","Tradiciones","Gastronomia",
4 "Platos Bandera","Contactenos"];
5
6 const items=Lista.map((x) =>
7   <a className="item" href="#">{x}</a>
8 );
9
10 function Menu( ){
11   return(
12     <div class="bloque">
13       |   {items}
14       |   </div>
15     );
16 }
17 export default Menu;

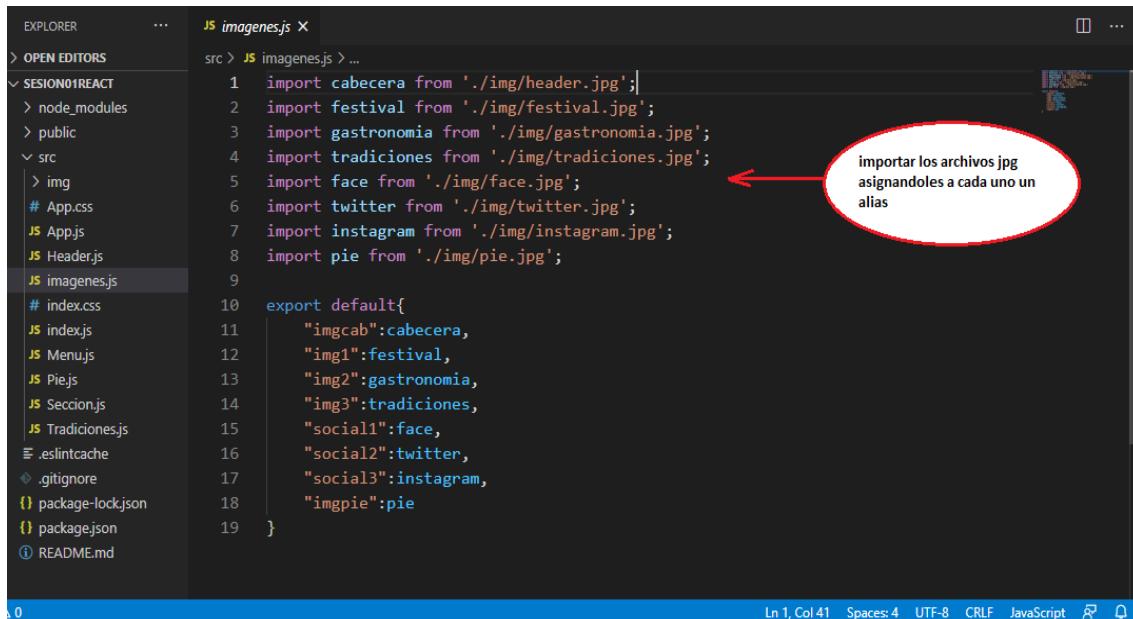
```

Figura 201 : Caso práctico

Fuente.- Elaboración Propia

### Trabajando con el archivo Header.js

Primero, En el archivo imágenes.js, vamos a importar la lista de imágenes del proyecto (ubicadas en la carpeta img) y la exportamos como una lista de objetos



```

EXPLORER      ...
OPEN EDITORS
src > JS imagenes.js > ...
1 import cabecera from './img/header.jpg';
2 import festival from './img/festival.jpg';
3 import gastronomia from './img/gastronomia.jpg';
4 import tradiciones from './img/tradiciones.jpg';
5 import face from './img/face.jpg';
6 import twitter from './img/twitter.jpg';
7 import instagram from './img/instagram.jpg';
8 import pie from './img/pie.jpg';
9
10 export default{
11   "imgcab":cabecera,
12   "img1":festival,
13   "img2":gastronomia,
14   "img3":tradiciones,
15   "social1":face,
16   "social2":twitter,
17   "social3":instagram,
18   "imgpie":pie
19 }

```

Figura 202: Caso práctico

Fuente.- Elaboración Propia

En el archivo Header.js, importamos React y el archivo imágenes.js. defina el componente Titulo(props) donde retorna el <h1> y el componente Imagen() donde retorna la imagen. La función Header, retorna el bloque que agrupa el Titulo() y la Imagen().

```

EXPLORER      ...
JS Header.js ×

src > JS Header.js > ...
1  import React from 'react';
2  import imagenes from './imagenes'; ← importar el archivo
3  function Titulo(props){
4      return(
5          <h1 className="h1cab">{props.nombre}</h1>
6      );
7  }
8
9  function Imagen(){
10     return(
11         <img className="imgcab"  src={imagenes.imgcab} />
12     );
13 }
14
15 function Header(){
16     return(
17         <div className="bloque" >
18             <Titulo nombre="Tradiciones Y Costumbres en el Perú"></Titulo>
19             <Imagen/></Imagen>
20         </div>
21     );
22 }
23
24 export default Header; ← export default Header;
25

```

Figura 203 :Caso práctico  
Fuente.- Elaboración Propia

### Trabajando con el archivo Seccion.js

Crea el archivo Tradiciones.js, el cual retorna (export default) la lista de imágenes definida desde el import.

```

EXPLORER      ...
JS Seccion.js   JS Tradiciones.js ×

src > JS Tradiciones.js > default
1  import rest1 from './img/tradiciones1.jpg';
2  import rest2 from './img/tradiciones2.jpg';
3  import rest3 from './img/tradiciones3.jpg';
4  import rest4 from './img/tradiciones4.jpg';
5  import rest5 from './img/tradiciones5.jpg';
6  import rest6 from './img/tradiciones6.jpg';
7  import rest7 from './img/tradiciones7.jpg';
8  import rest8 from './img/tradiciones8.jpg';
9  import rest9 from './img/tradiciones9.jpg';
10
11 export default[←
12     rest1,
13     rest2,
14     rest3,
15     rest4,
16     rest5,
17     rest6,
18     rest7,
19     rest8,
20     rest9
21 ]

```

Figura 204 : Caso práctico  
Fuente.- Elaboración Propia

En el archivo Seccion.js, defina la constante imgs que almacena la lista de <img> mapeando desde Tradiciones. En el componente Seccion envío el título <h2> y la lista de imgs.

```

EXPLORER      ...
OPEN EDITORS
SESION01REACT
src
  img
  # App.css
  JS App.js
  JS Header.js
  JS imagenes.js
  # index.css
  JS index.js
  JS Menu.js
  JS Pie.js
  JS Seccion.js
  JS Tradiciones.js
.eslintcache
.gitignore
package-lock.json
package.json
README.md

JS Seccion.js X
src > JS Seccion.js > ...
1 import React from 'react';
2 import Tradiciones from './Tradiciones';
3
4 const imgs=Tradiciones.map((x) =>
5   <img className="imgseccion" src={x}></img>
6 );
7
8 function Seccion(){
9   return(
10     <div className="divseccion">
11       <h2 className="h2cab">Tradiciones en el Perú</h2>
12       {imgs}
13     </div>
14   );
15 }
16
17 export default Seccion;

```

Figura 205 : Caso práctico  
Fuente.- Elaboración Propia

### Trabajando con Pie.js

En el archivo Pie.js, envió la imagen del pie de página definido por el componente Imgpie(). Para ello importa el archivo imágenes.js

```

EXPLORER      ...
OPEN EDITORS
SESION01REACT
src
  img
  # App.css
  JS App.js
  JS Header.js
  JS imagenes.js
  # index.css
  JS index.js
  JS Menu.js
  JS Pie.js
  JS Seccion.js
  JS Tradiciones.js
.eslintcache
.gitignore
package-lock.json
package.json
README.md

JS Pie.js X
src > JS Pie.js > Pie
1 import React from 'react';
2 import imagenes from './imagenes';
3
4 function Imgpie(){
5   return(
6     <img className="imgpie" src={imagenes.imgpie}></img>
7   );
8 }
9
10 function Pie(){
11   return(
12     <div>
13       <Imgpie></Imgpie>
14     </div>
15   )
16 }
17
18 export default Pie;

```

Figura 206 : Caso práctico  
Fuente.- Elaboración Propia

En la página Index.html (localhost:3000), se visualiza el contenido de la página.

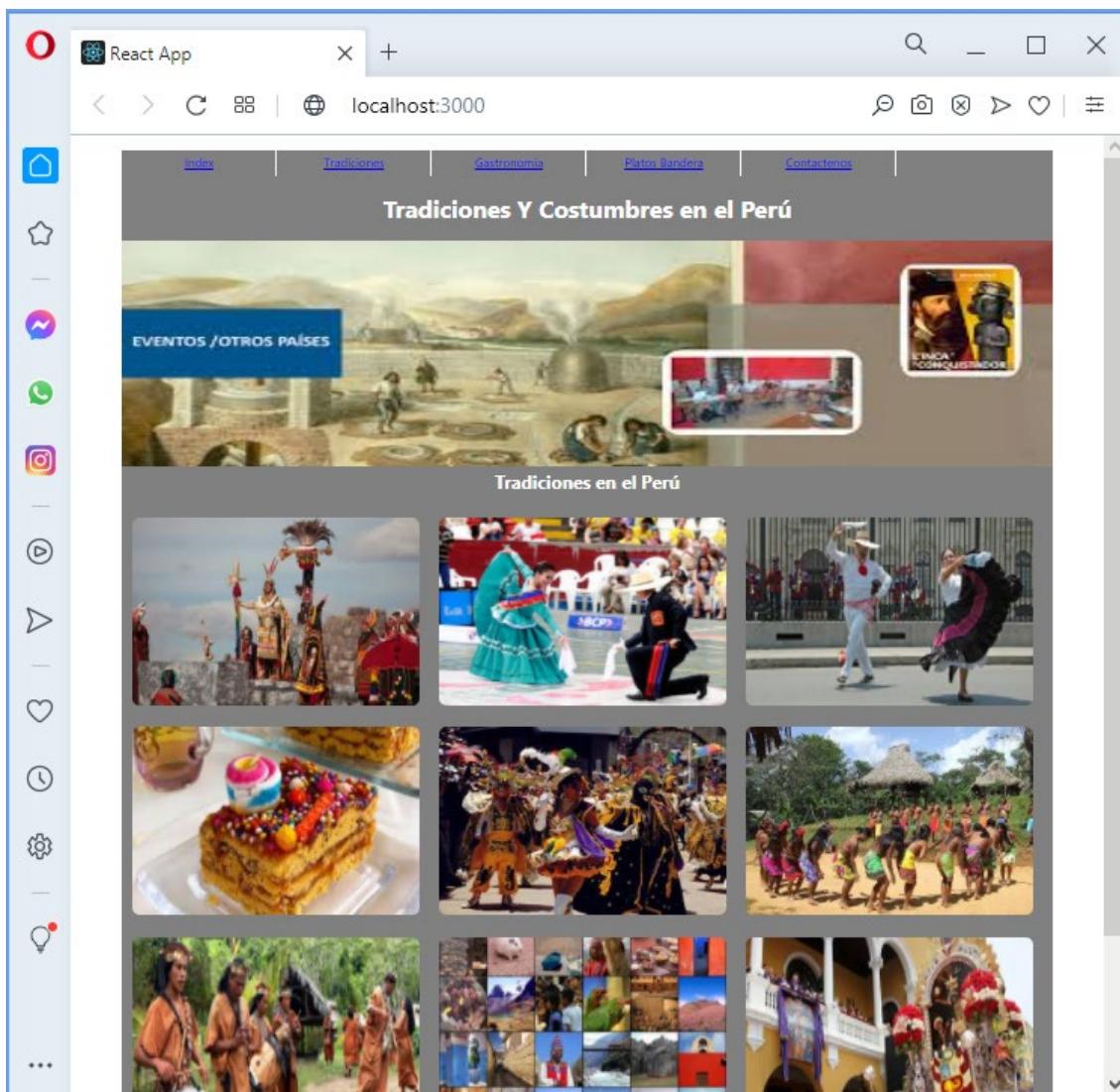


Figura 207 : Caso práctico

Fuente.- Elaboración Propia

# Resumen

1. React es una biblioteca escrita en JavaScript, desarrollada en Facebook para facilitar la creación de componentes interactivos, reutilizables, para interfaces de usuario.
2. Un componente en React es una interface de usuario el cual es reutilizable, son objetos los cuales se crean en JSX. Utiliza la misma sintaxis dentro de JavaScript XML con la misma sintaxis dentro de JavaScript. WebPack se encarga de convertir este código a JavaScript estándar que el navegador puede interpretar.
3. Existen dos tipos de componentes: componente funcionales que generan elementos React, y el componente de clase el cual tiene propiedades, ciclo de vida.
4. El Virtual DOM se basa en una idea bastante sencilla e ingeniosa. Básicamente hace que, cuando se actualiza una vista, React se encargue de actualizar el DOM Virtual, que es mucho más rápido que actualizar el DOM del navegador (DOM real).
5. Cuando React compara el DOM Virtual con el DOM del navegador sabe perfectamente qué partes de la página debe actualizar y se ahorra la necesidad de actualizar la vista entera.

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://es.reactjs.org/docs/create-a-new-react-app.html#create-react-app>
- <https://codingpotions.com/react-componentes>
- <https://es.reactjs.org/docs/components-and-props.html>
- <https://ed.team/blog/como-funciona-reactjs>
- <https://desarrolloweb.com/articulos/que-es-react-motivos-uso.html>
- <https://www.youtube.com/watch?v=zIY87vU33aA>
- <https://dev.to/izakntun/fundamentos-de-react-189e>
- <https://es.reactjs.org/docs/getting-started.html>

## 6.2. TRABAJANDO CON REACT

### Presentando JSX

JSX es una extensión de la sintaxis de JavaScript que permite describir cómo debería ser la interfaz de usuario. JSX produce “elementos” de React, los cuales no son requeridos, pero son útiles como ayuda visual cuando trabajas con interfaz de usuario en JavaScript.

React acepta el hecho de que la lógica de renderizado está intrínsecamente unida a la lógica de la interfaz de usuario: cómo se manejan los eventos, cómo cambia el estado con el tiempo y cómo se preparan los datos para su visualización.

En lugar de separar artificialmente tecnologías poniendo el maquetado y la lógica en archivos separados, React separa intereses con unidades ligeramente acopladas llamadas “**componentes**” que contienen ambas.

### Insertando expresiones en JSX

Puedes poner cualquier expresión de JavaScript dentro de llaves en JSX. Después de compilarse, las expresiones JSX se convierten en llamadas a funciones JavaScript regulares y se evalúan en objetos JavaScript. Esto significa que puedes usar JSX dentro de declaraciones if y bucles for, asignarlo a variables, aceptarlo como argumento, y retornarlo desde dentro de otras funciones.

```
const titulo=<h1>Cibertec</h1>

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

Figura 208: expresiones JSX  
Fuente.- Elaboración Propia

### Especificando hijos con JSX

Las etiquetas de JavaScript pueden contener bloques y nodos hijos

```
const bloque=(
  <div>
    <h1>Cibertec</h1>
    <img src={logocibertec}></img>
  </div>
);
```

Figura 209 : expresiones JSX  
Fuente.- Elaboración Propia

## JSX representa objetos

Babel compila JSX a llamadas de `React.createElement()`, el cual realiza algunas comprobaciones para escribir código libre de errores, pero, en esencia crea un objeto.



```

const titulo=React.createElement(
  'h1',
  {className:'h1cab'},
  'Hola Mundo'
);

ReactDOM.render(
  titulo,
  document.getElementById('root')
);

```

Figura 210: Create Element en JSX

Fuente.- Elaboración Propia

## Renderizando un elemento en el DOM

En nuestra página `index.html`, tenemos un `<div>` cuyo `id="root"`: `<div id="root"></div>`. Lo llamamos un nodo “raíz” porque todo lo que esté dentro de él será manejado por React DOM.

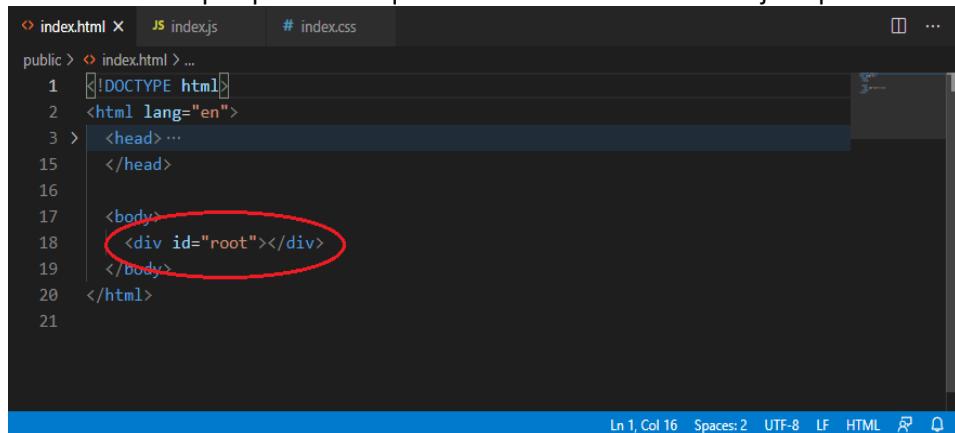


Figura 211: Renderizando un elemento

Fuente.- Elaboración Propia

Las aplicaciones construidas con React usualmente tienen un único nodo raíz en el DOM. Para renderizar un elemento de React en un nodo raíz del DOM, pasa ambos a `ReactDOM.render()`:

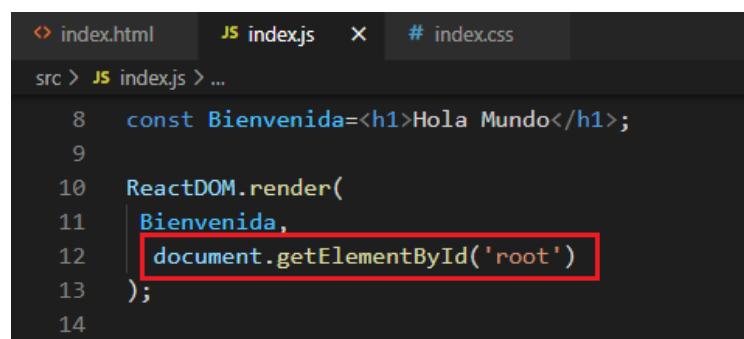


Figura 212: Renderizando un elemento

Fuente.- Elaboración Propia

### 6.2.1 Componentes react con createclass

Los componentes funcionales son componentes que generan elementos React.

Por convenio se pone el nombre de la función en mayúsculas. Para renderizarlo simplemente se pone una etiqueta con el nombre de la función.

Por ejemplo:

```
function HelloWorld(){
  return(
    <h1>Hola Mundo</h1>
  );
}

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

Figura 213 : Funciones  
Fuente.- Elaboración Propia

A las funciones podemos pasar parámetros, llamados propiedades, por ejemplo:

```
function HelloWorld(props){
  return(
    <div>
      <h1>Hola Mundo, {props.name}</h1>
      <h2>Curso: {props.curso}</h2>
    </div>
  );
}

ReactDOM.render(
  <HelloWorld name="Cibertec" curso="Entornos Web" />,
  document.getElementById('root')
);
```

Figura 214: Funciones y propiedades  
Fuente.- Elaboración Propia

En este ejemplo al renderizar el elemento, pasamos una o más variables al componente que se encargará de renderizarla. Puedes pasar como parámetros variables, arrays e incluso otros elementos React.

En este ejemplo utilizamos un array para definir una lista de hipervínculos los cuales son renderizados en la página index.html

```

const items=["Index","Carreras","Matriculas","Convenios","Contactenos"];

function Menu(){
  let vinculos=items.map((x) =>
    <a className="item" href="#">{x}</a>
  );
  return(
    <div>
      {vinculos}
    </div>
  );
}

function HelloWorld(props){ ... };

ReactDOM.render(
  <div>
    <Menu />
    <HelloWorld name="Cibertec" />
  </div>,
  document.getElementById('root')
);

```

Figura 215: Funciones y Arrays  
Fuente.- Elaboración Propia

## Creando elementos en React

Para crear un elemento, importamos los scripts de React y React-Dom, con ellos tenemos acceso a la API de react.

React.createElement recibe 3 parámetros básicos:

- El tipo de elemento html
- Las propiedades del elemento
- El contenido del elemento

Un elemento creado con react es un object con propiedades, que luego ReactDOM sabe renderizar en el DOM.

```

1 import React from 'react';
2 import ReactDOM from 'react-dom';
3
4 const bloque=React.createElement(
5   'div',
6   {className:'clase'},
7   'Hola Mundo'
8 );

```

Figura 216: Create Element  
Fuente.- Elaboración Propia

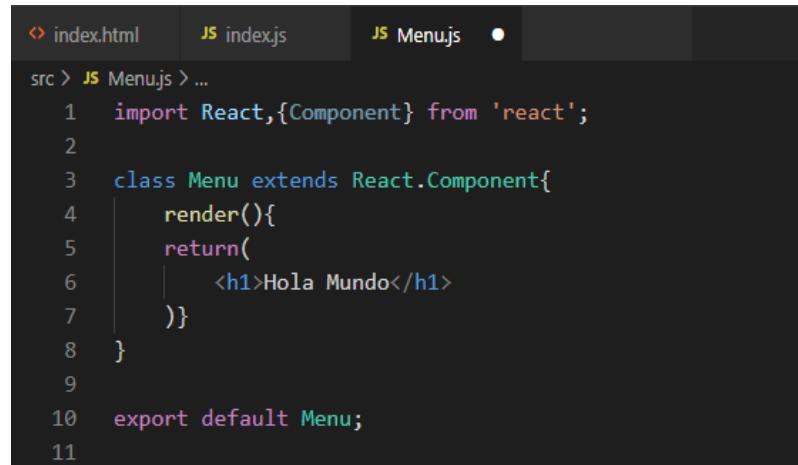
### 6.2.2. Componentes react mediante clases es6

Un componente en React es una pieza de UI (user interface), que tiene una funcionalidad independiente definida. Un componente se define usando clases, donde una de las formas es

utilizar ES6 o ECMAScript 6, la cual nos permite utilizar programación orientada a objetos (POO) para definir un componente.

Para que una clase se considere componente en React, mínimo debe:

- Extender de la clase React.Component
- Debe contar con un método de nombre render el cual debe retornar un elemento de React.



```

index.html JS index.js JS Menu.js

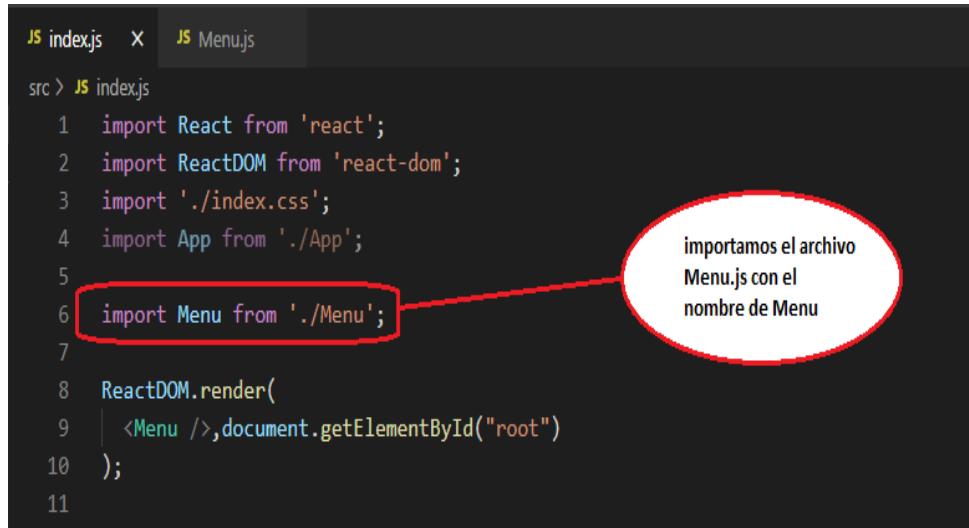
src > JS Menu.js > ...
1 import React,{Component} from 'react';
2
3 class Menu extends React.Component{
4     render(){
5         return(
6             <h1>Hola Mundo</h1>
7         )}
8 }
9
10 export default Menu;
11

```

Figura 217: Componente de clase  
Fuente.- Elaboración Propia

En el ejemplo anterior, hemos creado una clase Menu que extiende de la clase base React.Component. En nuestra clase definimos el método render(), este método es obligatorio y nativo de React en el cual devolvemos un elemento <h1>.

Definido nuestro componente, a continuación, lo incluimos en el DOM y para ello utilizamos el método ReactDOM.render.



```

JS index.js X JS Menu.js

src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5
6 import Menu from './Menu';
7
8 ReactDOM.render(
9     <Menu />,document.getElementById("root")
10 );
11

```

Figura 218: Componente de clase  
Fuente.- Elaboración Propia

En este segundo ejemplo, hemos definido la clase Navega, el cual define dos variables y retorna el contenido de vínculos, junto con el valor de retorno de <Menu />. Como pueden observar, la clase puede realizar operaciones dentro del cuerpo de este.

```

JS index.js      JS Menu.js      JS Navega.js X
src > JS Navega.js > ...
1 import React,{Component} from 'react';
2 import Menu from './Menu';
3
4 class Navega extends React.Component{
5     render(){
6         const items=["Index","Carreras","Matriculas","Convenios","Contactenos"];
7         let vinculos=items.map((x) =>
8             | | | <a className="item" href="#">{x}</a>
9         );
10        return(
11            <div>
12                <Menu/>
13                {vinculos}
14            </div>
15        )
16    }
17
18 export default Navega;
19
20

```

importar React y el archivo Menu.js

retorno el valor de la clase Menu y la variable {vinculos} el cual almacena un Array de hipervínculos

Figura 219 : Componente de clase  
Fuente.- Elaboración Propia

### 6.2.3. Propiedades y estados en componentes react

Uno de los recursos elementales que tenemos al desarrollar componentes React son las **propiedades**.

Las propiedades sirven para cargar de datos a los componentes, de modo que éstos puedan personalizar su comportamiento, o para transferir datos de unos componentes a otros para producirse la interoperabilidad entre componentes. Existen diversas maneras de pasar propiedades, aunque lo más común es hacerlo de manera declarativa, al usar el componente dentro del código JSX.

#### Propiedades en componentes definidos mediante clases

Cuando definimos clases, las propiedades enviadas al componente, las recibimos en forma de propiedades o atributos del objeto. Por tanto, accedemos a ellas a través de "**this**". En concreto usaremos **this.props** para acceder a un objeto donde tenemos todas las propiedades definidas como atributos.

Nuestro componente "Bienvenido" le pasamos dos propiedades "nombre" y "curso". Entonces podremos usar esas propiedades de la siguiente manera:

```

JS index.js      JS Bienvenido.js X
src > JS Bienvenido.js > ...
1 import React,{Component} from 'react';
2
3 class Bienvenido extends React.Component{
4   render(){
5     return(
6       <div>
7         <h1>Hola {this.props.nombre}, bienvenido al curso {this.props.curso}</h1>
8       </div>
9     )
10  }
11
12 export default Bienvenido;
13

```

Figura 220 : Propiedades de clase

Fuente.- Elaboración Propia

Para enviar valores a las propiedades, se hace de forma declarativa en el código JSX, cuando usamos un componente.

Cada propiedad se indica como si fuera un atributo del HTML, al que le indicamos su valor.

```

JS index.js X  JS Bienvenido.js
src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5
6 import Bienvenido from './Bienvenido';
7
8 ReactDOM.render(
9   <Bienvenido nombre="Carlos" curso="Desarrollo de Entornos Web"/>,
10  document.getElementById("root")
11 );
12
13

```

Figura 221: Propiedades de clase

Fuente.- Elaboración Propia

### Propiedades en componentes generados a partir de funciones

Otro modo de definir este tipo de componentes sencillos es por medio de simples funciones, lo que dulcifica bastante la sintaxis de implementación de un componente. En este caso las propiedades se recibirán por parámetro en la función que implementa el componente.

```

JS index.js      JS Bienvenido.js X
src > JS Bienvenido.js > ...
1 import React,{Component} from 'react';
2
3 function Bienvenido(propiedades){
4   return(
5     <div>
6       <h1>Hola [propiedades.nombre], bienvenido al curso [propiedades.curso]</h1>
7     </div>
8   )
9 }
10
11 export default Bienvenido;
12

```

Figura 222: Propiedades de funciones

Fuente.- Elaboración Propia

Al definir la función se prescinde del método render, porque no estamos haciendo una clase. La propia función es el equivalente al método render() que teníamos al crear componentes por medio de una clase ES6. Por lo tanto, devuelve el JSX para representar el componente.

### Estados en un componente React

Los componentes **sin estado** no guardan ninguna información y por ello no necesitan de datos locales. Eso no significa que no puedan personalizar su comportamiento, lo que se consigue con la ayuda de las propiedades de los componentes. Estas propiedades que nos pasan se podrán incluso transformar al producir una salida, de modo que sea acorde con las necesidades, pero no se guardará ningún valor y el componente no tendrá un ciclo de vida.

```

JS index.js      JS Bienvenido.js X
src > JS Bienvenido.js > ...
1 import React,{Component} from 'react';
2
3 function Bienvenido(propiedades){
4   var date = new Date();
5   var fecha = date.getDate() + '/' + (date.getMonth() + 1) + '/' + date.getFullYear();
6   return(
7     <div>
8       <h1>Hola [propiedades.nombre], bienvenido al curso [propiedades.curso]</h1>
9       <h2>Hoy es [fecha]</h2>
10    </div>
11  )
12 }
13
14 export default Bienvenido;
15

```

Figura 223: Componente sin estado

Fuente.- Elaboración Propia

El estado de un componente es la memoria en cada momento que tiene la instancia de un componente que se está mostrando en pantalla. Se trata de un atributo de la instancia parecido a las props, al que podremos acceder con this.state.

Al contrario que las props, el estado varía durante el tiempo en que el componente aparece pintado en la pantalla. Es decir, las props no podemos cambiarlas, pero los estados, sí, aunque de cierta manera.

```

JS index.js JS Bienvenido.js ...
src > JS Bienvenido.js > ...
1 import React,{Component} from 'react';
2
3 const imagenes=["/img/img1.jpg","/img/img2.jpg","/img/img3.jpg",
4 "/img/img4.jpg","/img/img5.jpg","/img/img6.jpg","/img/img7.jpg"];
5 class Bienvenido extends Component{
6   constructor(props){
7     super(props);
8     this.state={
9       c:Math.floor(Math.random(imagenes.length)*7),
10      curso:"Desarrollo de Entornos Web"
11    };
12  };
13  state={
14    c:0,
15    curso:""
16  };
17  render(){
18    return(
19      <div>
20        <h1>Bienvenido al curso {this.state.curso}</h1>
21        <img src={imagenes[this.state.c]} />
22      </div>
23    );
24  }
25 }
26
27 export default Bienvenido;
28

```

Ln 30, Col 1 Spaces:4 UTF-8 CRLF JavaScript ⚡ ⚡

asignar valores iniciales a las variables c y curso

declarar las variables c y curso en state

Figura 224 : Estado de un Componente

Fuente.- Elaboración Propia

En la clase definimos un constructor, el cual inicializa la propiedad "state" del componente. Como puedes ver el estado es un objeto, en el que ponemos tantos atributos como sea necesarios guardar como estado.

A la hora de renderizar el componente, por supuesto, podremos usar el estado para mostrar la salida. En este caso puedes ver cómo se vuelca el estado en la vista, con la expresión {this.state.curso}.

```
JS index.js JS Bienvenido.js X
src > JS Bienvenido.js > ...
6   class Bienvenido extends Component{
7     >     constructor(props){ ...
14   };
15   state={
16     c:0,
17     curso:""
18   };
19
20   ciclo(){
21     this.setState({
22       c:Math.floor(Math.random(imagenes.length)*7)
23     });
24   };
25
26   render(){
27     return(
28       <div>
29         <h1>Bienvenido al curso {this.state.curso}</h1>
30         <img src={imagenes[this.state.c]} />
31         <button onClick={this.ciclo.bind(this)}>Hazme Click</button>
32       </div>
33     );
34   }
35
36   export default Bienvenido;
37
```

Figura 225: Uso del setState()

Fuente.- Elaboración Propia

Para modificar el estado de un componente no debemos utilizarlo con la propiedad `this.state`, sino que tenemos que hacerlo a través de `this.setState()`.

El motivo es que `setState()`, además de alterar el estado, desencadena toda una serie de acciones implementadas en el core de React, que se encargan de realizar todo el trabajo por debajo para que ese cambio de estado tenga una representación en la vista.

Dicho de otra manera, en el momento que cambiamos el estado con `setState()`, se pone en ejecución el motor de React para que se actualice el DOM virtual, se compare con el DOM del navegador y por último se actualicen aquellos elementos que sea necesario (porque realmente hayan cambiado). Si no usamos `setState()` todas esas operativas no se producirían y los componentes empezarían a funcionar de manera no deseada.

## LABORATORIO 1

### Trabajando con componentes en una aplicación React JS

En este laboratorio vamos a implementar la página index.html utilizando componentes y manejando estado de éstos.

#### 1. Trabajando con el componente Menu.js

##### Agregando un selector a index.css

Primero agregamos en el archivo index.css, el selector de class ítem y definimos sus propiedades. Consideraremos que agregamos 6 hipervínculos.



```
# index.css X
src > # index.css > ↗ h1
19
20   .item{
21     width: 16.6%;
22     height: 30px;
23     padding-top: 10px;
24     text-align: center;
25     display: inline-block;
26     color: white;
27     background-color: gray;
28   }
29
30   .item:hover{
31     color: gray;
32     background-color: white;
33   }
34
```

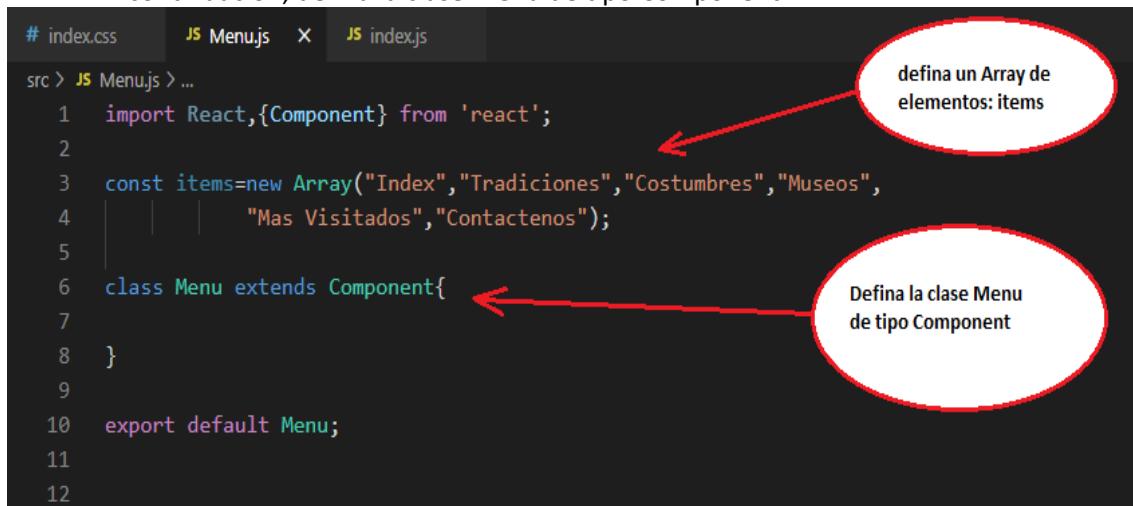
Figura 226: index.css

Fuente.- Elaboración Propia

##### Trabajando con Menu.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Menu.js.

- En el archivo importar React y {Component}
- Defina un Array llamado ítems, el cual almacena la lista de elementos (tipo cadena)
- A continuación, defina la clase Menu de tipo Component



```
# index.css JS Menujs X JS index.js
src > JS Menujs > ...
1 import React,{Component} from 'react';
2
3 const items=new Array("Index","Tradiciones","Costumbres","Museos",
4           "Mas Visitados","Contactenos");
5
6 class Menu extends Component{
7
8 }
9
10 export default Menu;
11
12
```

Figura 227: Menu.js

Fuente.- Elaboración Propia

En la clase Menu:

- Defina la componente lista de tipo Array, state
- En el constructor, inicialice el valor de lista con un Array de hipervínculos <a>
- Al renderizar, envío el valor de la componente lista: {this.state.lista}

```

# index.css      JS Menu.js  X  JS index.js
src > JS Menu.js > ...
6  class Menu extends Component{
7
8    constructor(props){
9      super(props);
10     this.state={
11       lista:items.map((it) => <a className="item" href="#">{it}</a>)
12     }
13   };
14
15   state={
16     lista:new Array()
17   };
18
19   render(){
20     return(
21       <div>{this.state.lista}</div>
22     );
23   };
24 }
25
26 export default Menu;
27

```

Figura 228: Menu.js  
Fuente.- Elaboración Propia

### Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar el archivo Menu.js y lo llamaremos Menu
- Al renderizar la página importamos el contenido de Menu

```

# index.css      JS Menu.js  X  JS index.js
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5
6  import Menu from './Menu';
7
8  ReactDOM.render(
9    <Menu/>,
10   document.getElementById("root")
11 );
12

```

Figura 229: Index.js  
Fuente.- Elaboración Propia

En la ventana de consola, ejecutar el proyecto React JS: **npm start sesion02react** y dar Enter, donde se visualiza el contenido de Menu.js

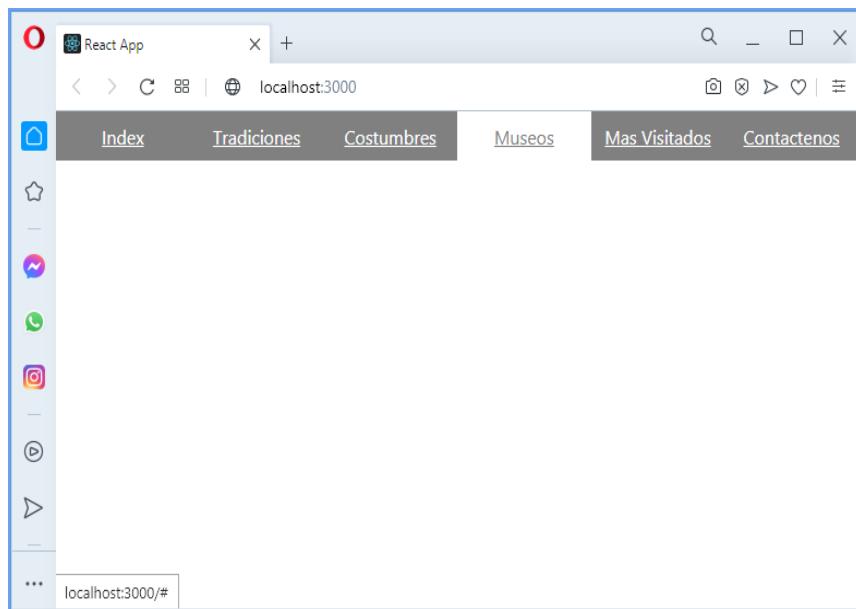


Figura 230 : Index.html

Fuente.- Elaboración Propia

## 2. Trabajando con el componente Header.js

### Agregando selectores a index.css

Primero agregamos en el archivo index.css, los selectores de class container, h1cab, imgcab y definimos sus propiedades.

```
# index.css X JS index.js JS Header.js
src > # index.css > code
29
30 .container{
31   width: 90%;
32   height: auto;
33   margin-top: 10px;
34   margin-bottom: 10px;
35   margin-left: 5%;
36   margin-right: 5%;
37 }
38
39 .h1cab{
40   color: gray;
41   text-align: center;
42 }
43 .imgcab{
44   padding: 3%;
45   background-color: gray;
46   width: 93%;
47   border: 1px solid black;
48   height: 350px;
49 }
```

definiendo el selector container (para el bloque principal)

definiendo el selector h1cab para el titulo del Header

definiendo el selector imgcab para la imagen del Header

Figura 231 : Index.css

Fuente.- Elaboración Propia

### Trabajando con Header.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Header.js.

- En el archivo importar React y {Component}
- Defina un Array llamado imagenes, el cual almacena la lista de direcciones de las imágenes las cuales se ubican en la carpeta img direccionada en public.
- A continuación, defina la clase Header de tipo Component

```

JS index.js   JS Header.js X
src > JS Header.js > ...
1 import React,{Component} from 'react';
2
3 const imagenes=["/img/img1.jpg","/img/img2.jpg","/img/img3.jpg",
4 "/img/img4.jpg","/img/img5.jpg","/img/img6.jpg","/img/img7.jpg"];
5
6 class Header extends Component{
7
8 }
9
10 export default Header;
11

```

Figura 232: Header.js  
Fuente.- Elaboración Propia

En la clase Header:

- En el constructor, inicialice el valor del estado “c” a cero
- Al renderizar, ejecuta un setTimeout() donde, por cada segundo, actualizo el valor de “c” con un valor aleatorio entre 0 hasta 7 (longitud del array imágenes).

```

JS index.js   JS Header.js X
src > JS Header.js > ...
>
6 class Header extends Component{
7     constructor(props){
8         super(props);
9         this.state={
10             c:0,
11         };
12     };
13     state={
14         c:0
15     };
16     render(){
17         setTimeout(() => this.setState({
18             c: Math.floor(Math.random()*imagenes.length)
19         }),1000);
20         return(
21             <div>
22                 <h1 className="h1cab">Un Viaje a las tradiciones del Perú</h1>
23                 <img className="imgcab" src={imagenes[this.state.c]} />
24             </div>
25         );
26     }
27
28     export default Header;
29

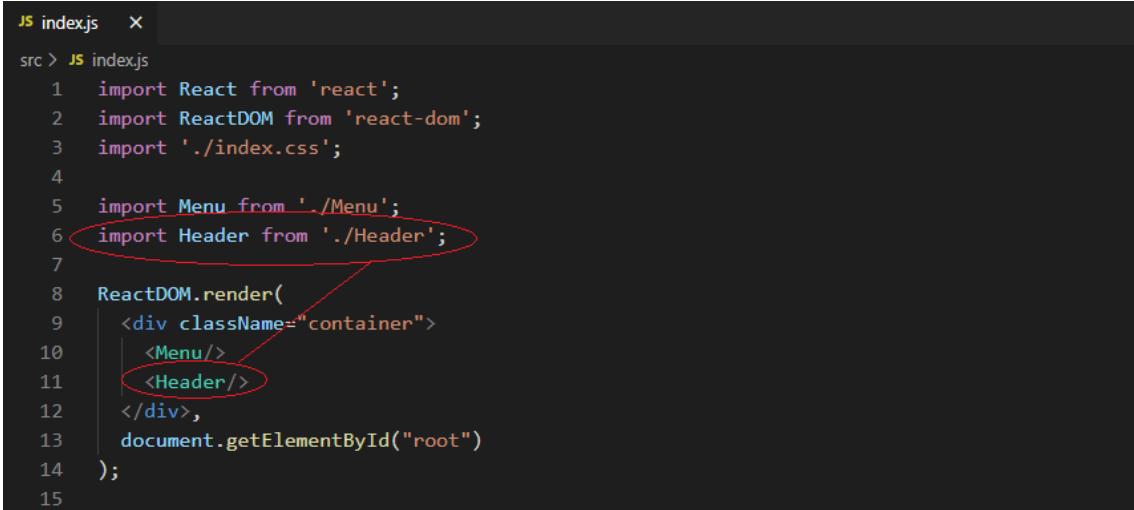
```

Figura 233 : Header.js  
Fuente.- Elaboración Propia

### Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar los archivos Menu.js y Header.js
- Al renderizar la página importamos el contenido de Menu y Header, tal como se muestra



```
JS index.js  X
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4
5  import Menu from './Menu';
6  import Header from './Header'; // Line 6 circled in red
7
8  ReactDOM.render(
9    <div className="container">
10      <Menu/>
11      <Header> // Line 11 circled in red
12    </div>,
13    document.getElementById("root")
14  );
15
```

Figura 234: index.js  
Fuente.- Elaboración Propia

Actualizar la página para visualizar el contenido de la class Header.

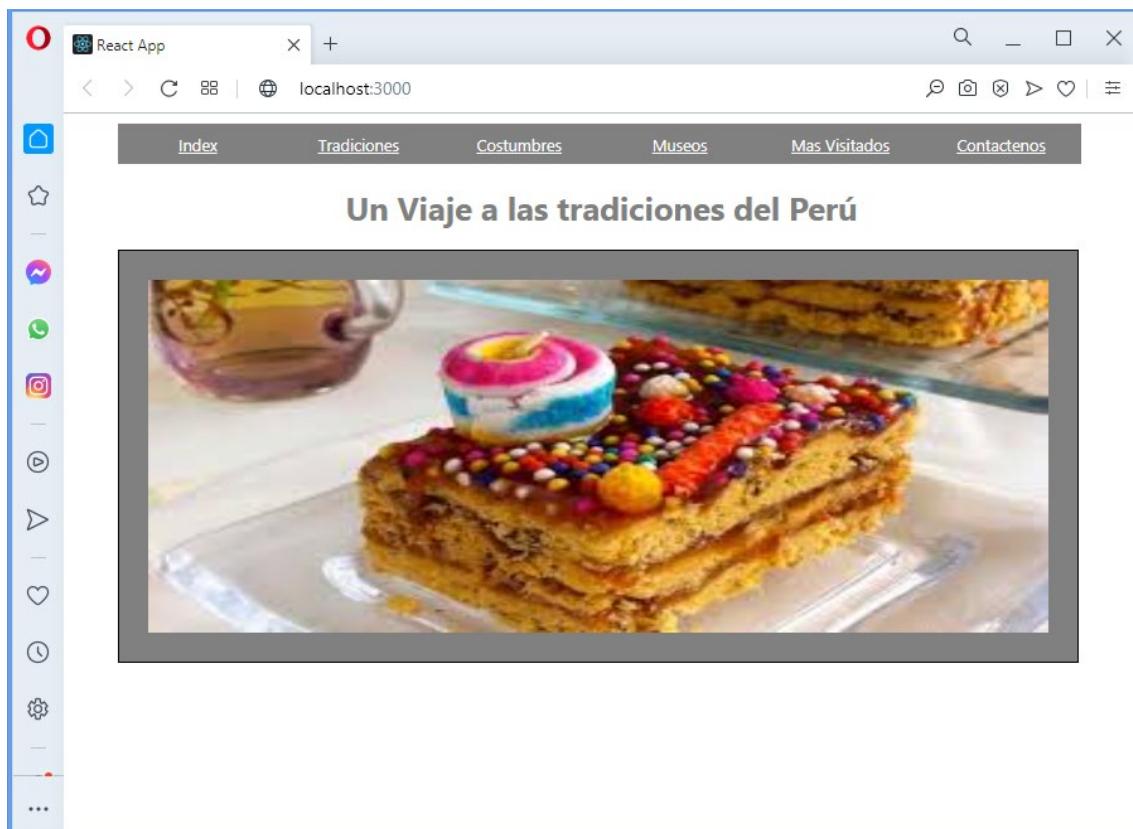


Figura 235 : Index.html  
Fuente.- Elaboración Propia

### 3. Trabajando con el componente RegionFecha.js

#### Agregando selectores a index.css

Primero agregamos en el archivo index.css, el selector de class div-fecha, defina sus propiedades

```

JS index.js      # index.css X JS RegionFecha.js
src > # index.css > .container
49
50   .div-fecha{
51     width: 50%;
52     display: inline-block;
53     height: 30px;
54     background-color: gray;
55     color: white;
56   }

```

Figura 236: index.css  
Fuente.- Elaboración Propia

### Trabajando con RegionFecha.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos RegionFecha.js.

- En el archivo importar React y {Component}
- Defina una variable f que es la instancia de la clase Date()
- A continuación, defina la clase RegionFecha de tipo Component

```

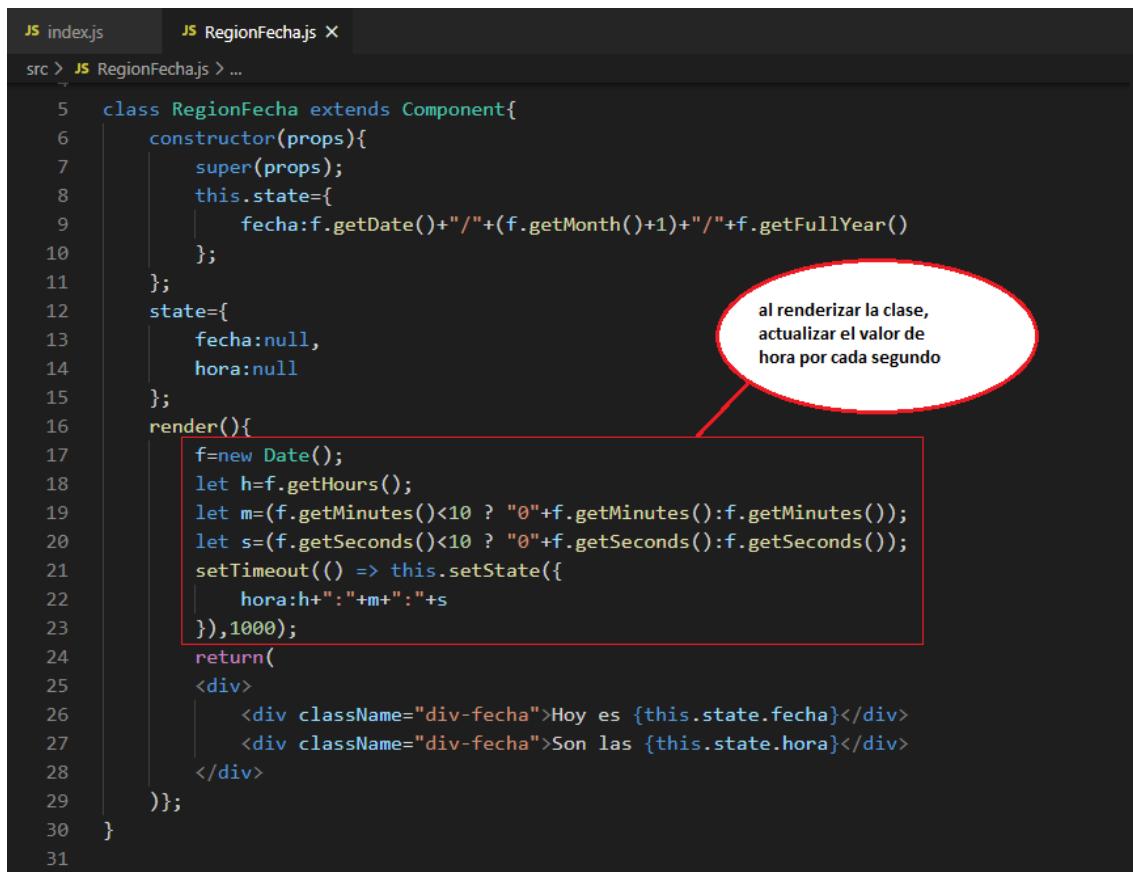
JS index.js      JS RegionFecha.js X
src > JS RegionFecha.js > ...
1  import React,{Component} from 'react';
2
3  let f=new Date(); ←
4
5  class RegionFecha extends Component{
6    ←
7  }
8
9  export default RegionFecha;
10

```

Figura 237 : RegionFecha.js  
Fuente.- Elaboración Propia

En la clase RegionFecha:

- Defina en el state, los estados fechan y hora
- En el constructor, inicialice el valor del estado “fecha” a la fecha del sistema
- Al renderizar, ejecuta un setTimeout() donde, por cada segundo, actualizo el valor de “hora” con los valores de hora (h), minutos (m) y segundos (s) concatenados.



```

JS index.js   JS RegionFecha.js X
src > JS RegionFecha.js ...
5  class RegionFecha extends Component{
6    constructor(props){
7      super(props);
8      this.state={
9        fecha:f.getDate()+(f.getMonth()+1)+f.getFullYear()
10     };
11    };
12    state={
13      fecha:null,
14      hora:null
15    };
16    render(){
17      f=new Date();
18      let h=f.getHours();
19      let m=(f.getMinutes()<10 ? "0"+f.getMinutes():f.getMinutes());
20      let s=(f.getSeconds()<10 ? "0"+f.getSeconds():f.getSeconds());
21      setTimeout(() => this.setState({
22        hora:h+":"+m+":"+s
23      }),1000);
24      return(
25        <div>
26          <div className="div-fecha">Hoy es {this.state.fecha}</div>
27          <div className="div-fecha">Son las {this.state.hora}</div>
28        </div>
29      );
30    }
31  }

```

al renderizar la clase, actualizar el valor de hora por cada segundo

Figura 238: RegionFecha.js

Fuente.- Elaboración Propia

## Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar los archivos Menu.js, Header.js y RegionFecha.js
- Al renderizar la página importamos el contenido de Menu, Header y RegionF, tal como se muestra



```

JS index.js X   JS RegionFecha.js
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4
5  import Menu from './Menu';
6  import Header from './Header';
7  import RegionF from './RegionFecha'
8
9  ReactDOM.render(
10    <div className="container">
11      <Menu/>
12      <Header/>
13      <RegionF/>
14    </div>,
15    document.getElementById("root")
16  );
17

```

Figura 239 : index.js

Fuente.- Elaboración Propia

Actualizar la página para visualizar el contenido de la class RegionFecha.



Figura 240 : Index.html  
Fuente.- Elaboración Propia

#### 4. Trabajando con el componente Tradiciones.js

##### Agregando selectores a index.css

Primero agregamos en el archivo index.css, el selector de class div-fecha, defina sus propiedades

```

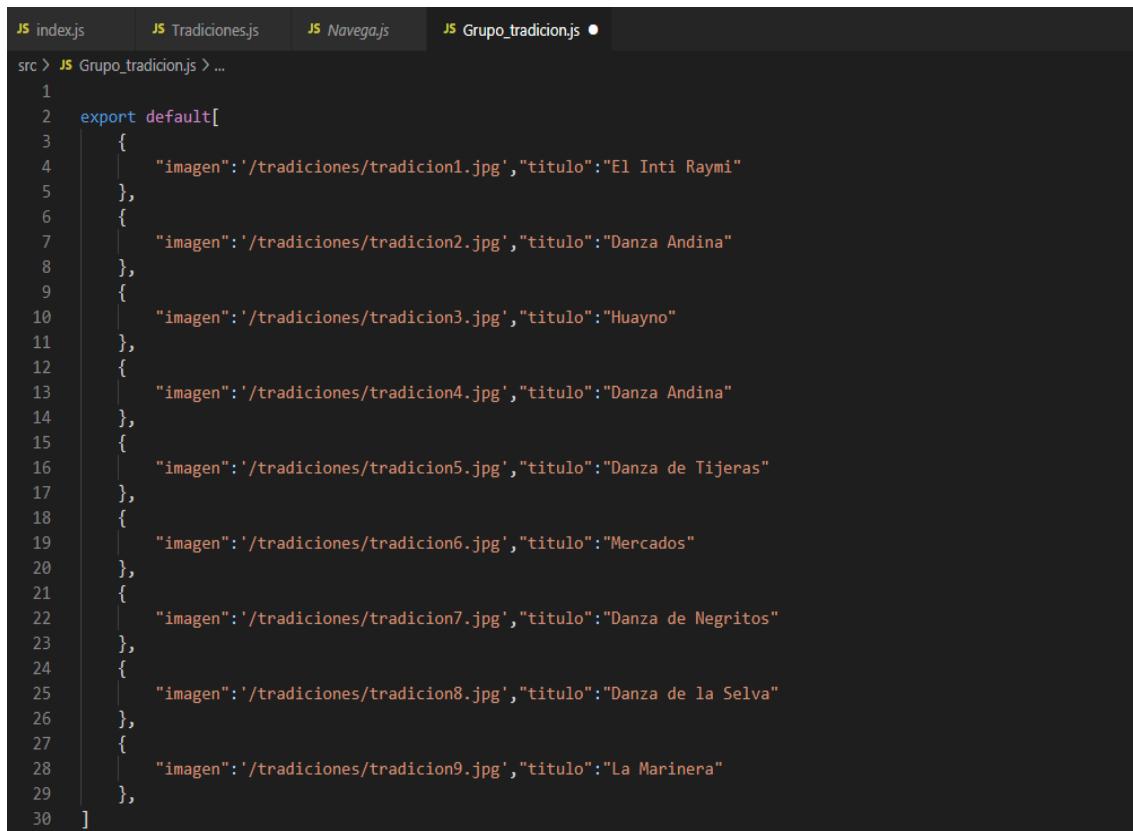
JS index.js      JS Tradiciones.js    # index.css X JS Navega.js      JS Grupo_tradicion.js
src > # index.css > h1cab
57
58 .art-tradicion{
59   width: 31%;
60   height: 250px;
61   padding: 1px;
62   border: 1px solid;
63   border-radius: 10px;
64   overflow: hidden;
65   display: inline-block;
66 }
67
68 .img-tradicion{
69   width: 100%;
70   height: 250px;
71 }
72
73 .h3cab{
74   text-align: center;
75   color: gray;
76 }
77
78 .art-tradicion:hover .img-tradicion{
79   height: 0px;
80   transition: all .05s;
81 }

```

Figura 241: index.css  
Fuente.- Elaboración Propia

### Definiendo un Array con Grupo\_tradicion.js

En el archivo exportamos un Array de elementos, el cual está conformado (para cada ítem), imagen y titulo, cada uno con su respectivo valor



```

JS index.js      JS Tradiciones.js      JS Navega.js      JS Grupo_tradicion.js •
src > JS Grupo_tradicion.js > ...
1
2  export default[
3  {
4    "imagen":'/tradiciones/tradiciones1.jpg',"titulo":"El Inti Raymi"
5  },
6  {
7    "imagen":'/tradiciones/tradiciones2.jpg',"titulo":"Danza Andina"
8  },
9  {
10   "imagen":'/tradiciones/tradiciones3.jpg',"titulo":"Huayno"
11 },
12 {
13   "imagen":'/tradiciones/tradiciones4.jpg',"titulo":"Danza Andina"
14 },
15 {
16   "imagen":'/tradiciones/tradiciones5.jpg',"titulo":"Danza de Tijeras"
17 },
18 {
19   "imagen":'/tradiciones/tradiciones6.jpg',"titulo":"Mercados"
20 },
21 {
22   "imagen":'/tradiciones/tradiciones7.jpg',"titulo":"Danza de Negritos"
23 },
24 {
25   "imagen":'/tradiciones/tradiciones8.jpg',"titulo":"Danza de la Selva"
26 },
27 {
28   "imagen":'/tradiciones/tradiciones9.jpg',"titulo":"La Marinera"
29 },
30 ]

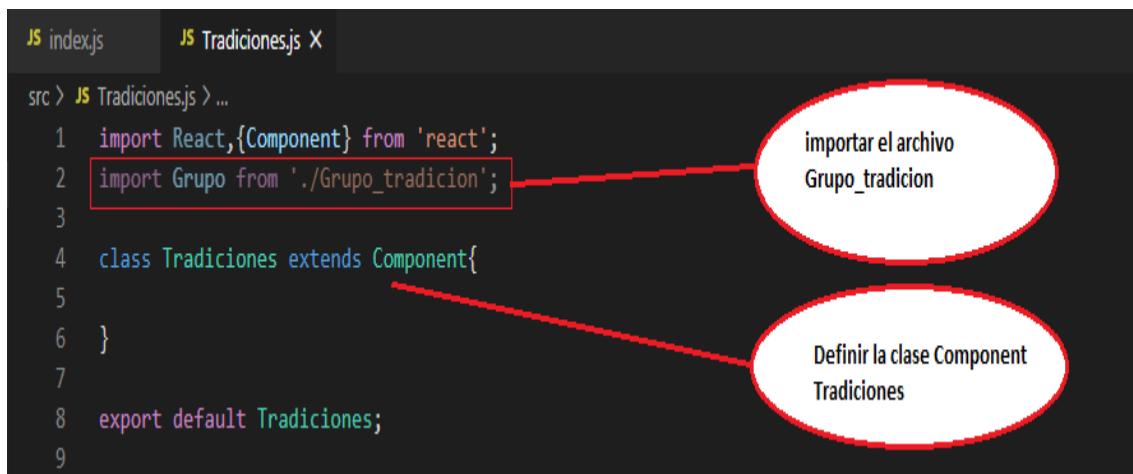
```

Figura 242: Grupo\_tradiciones  
Fuente.- Elaboración Propia

### Trabajando con Tradiciones.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Tradiciones.js.

- En el archivo importar React y {Component}, importar el archivo Grupo.
- A continuación, defina la clase RegionFecha de tipo Component



```

JS index.js      JS Tradiciones.js X
src > JS Tradiciones.js > ...
1 import React,{Component} from 'react';
2 import Grupo from './Grupo_tradiciones';
3
4 class Tradiciones extends Component{
5
6 }
7
8 export default Tradiciones;
9

```

Figura 243 : Tradiciones.js  
Fuente.- Elaboración Propia

En la clase Tradiciones:

- Defina en el state la lista
- En el constructor, inicialice la lista almacenando una colección de elementos leyendo el contenido de Grupo
- Al renderizar, visualizamos el contenido de lista

```

JS index.js JS Tradiciones.js X
src > JS Tradiciones.js > ...
4  class Tradiciones extends Component{
5    constructor(props){
6      super(props);
7      this.state={
8        lista:Grupo.map((x) =>
9          <article className="art-tradicion">
10         <img className="img-tradicion" src={x.imagen}></img>
11         <h3 className="h3cab">x.titulo</h3>
12       </article>
13     )
14   }
15 };
16 state={
17   lista:null
18 };
19 render(){
20   return(
21     <div>
22       <h1 className="h1cab">Nuestras Tradiciones</h1>
23       {this.state.lista}
24     </div>
25   )
26 }
27 }
28 export default Tradiciones;
29

```

Figura 244 : Tradiciones.js

Fuente.- Elaboración Propia

### Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar los archivos Menu.js, Header.js, RegionFecha.js y Tradiciones.js
- Al renderizar la página importamos el contenido de Menu, Header, RegionF y Tradiciones, tal como se muestra

```

JS index.js X
src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4
5 import Menu from './Menu';
6 import Header from './Header';
7 import RegionF from './RegionFecha';
8 import Tradiciones from './Tradiciones';
9
10 ReactDOM.render(
11   <div className="container">
12     <Menu/>
13     <Header/>
14     <RegionF/>
15     <Tradiciones />
16   </div>,
17   document.getElementById("root")
18 );
19

```

Figura 245 : index.js

Fuente.- Elaboración Propia

Actualizar la página para visualizar el contenido de la class Tradición.

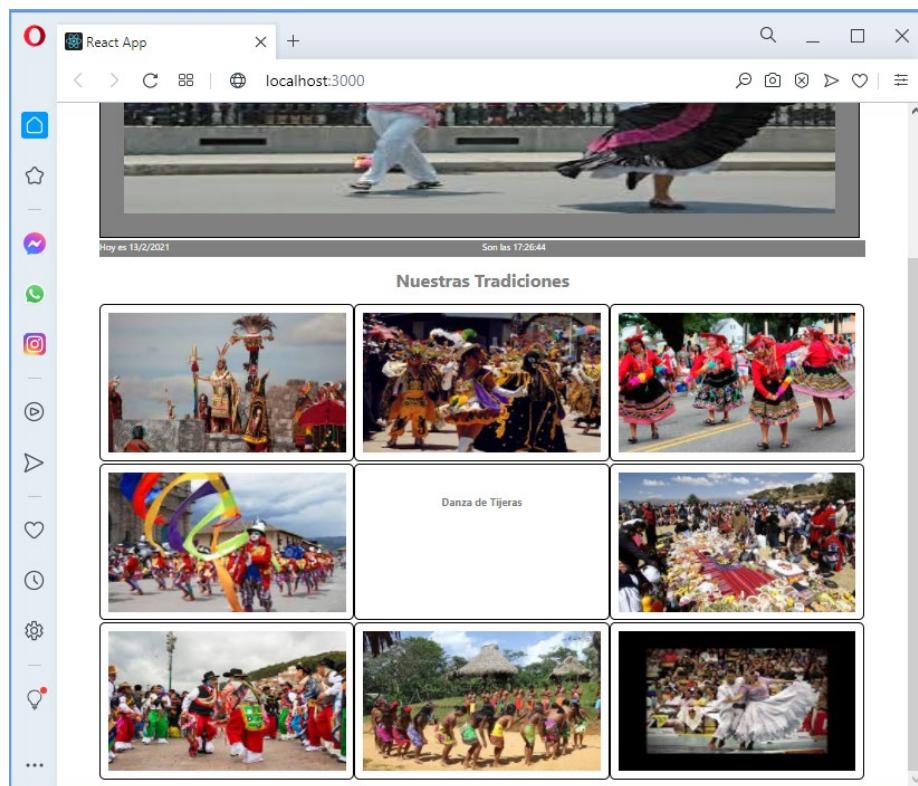


Figura 246: Index.html  
Fuente.- Elaboración Propia

## 5. Trabajando con el componente Footer.js

### Agregando selectores a index.css

Primero agregamos en el archivo index.css, el selector de class div-footer, img-rs y li-footer, defina sus propiedades.

```
# index.css X JS Footerjs JS index.js
src > # index.css > ↗ .img-tradicion
82
83 .div-footer{
84   width: 50%;
85   float: left;
86   min-height: 200px;
87   height: auto;
88   background-color: gray;
89   color: white;
90 }
91
92 .img-rs{
93   width: 40px;
94   padding-left: 5px;
95   height: 40px;
96 }
97
98 .li-footer {
99   margin: 0 20px;
100  list-style-type: none;
101  padding: 0;
102 }
```

definir el selector de class: div-footer

definir el selector de class para las imágenes: img-rs

definir el selector de class para los items: li-footer

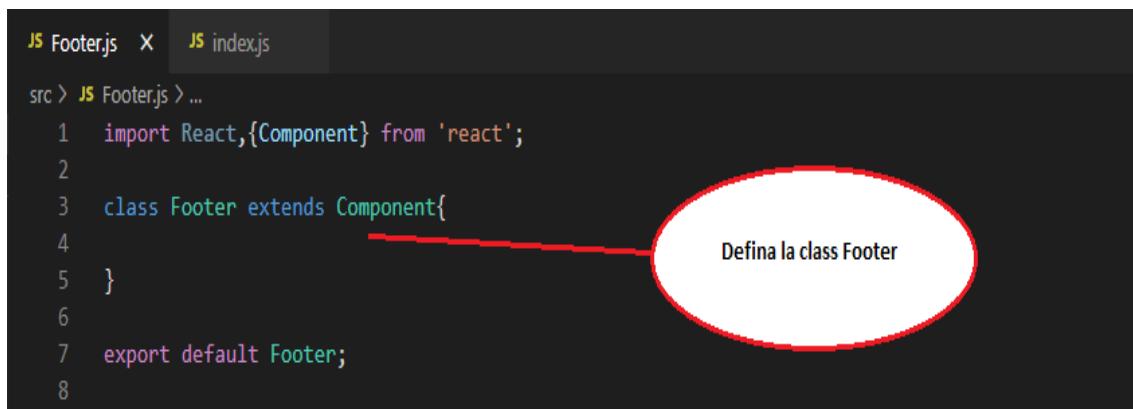
Figura 247: Index.css  
Fuente.- Elaboración Propia

### Trabajando con Footer.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Footer.js.

En el archivo importar React y {Component}

A continuación, defina la clase Footer de tipo Component



```

JS Footer.js X JS index.js
src > JS Footer.js > ...
1 import React,{Component} from 'react';
2
3 class Footer extends Component{
4
5 }
6
7 export default Footer;
8

```

Figura 248: Tradiciones.js  
Fuente.- Elaboración Propia

Defina los Arrays para listar las imágenes de las redes sociales (imgrs) y un segundo Array el cual almacena por cada ítem, un texto o descripción y route (dirección url)



```

JS Footer.js X JS index.js
src > JS Footer.js > Footer
1 import React,{Component} from 'react';
2
3 const imgrs=new Array("/img/face.jpg","/img/twitter.jpg","/img/instagram.jpg");
4
5 const listaOpciones = [
6   { text: "Index", route: "#" },
7   { text: "Tradiciones", route: "#" },
8   { text: "Costumbres", route: "#" },
9   { text: "Platos Tipicos", route: "#" },
10  { text: "Restaurantes", route: "#" },
11  { text: "Museos y Parques", route: "#" },
12  { text: "Calendario", route: "#" },
13  { text: "Contactenos", route: "#" }
14 ];
15
16 class Footer extends Component{
17
18 }
19
20 export default Footer;

```

Figura 249: Tradiciones.js  
Fuente.- Elaboración Propia

A continuación, defina:

- El objeto rs, el cual almacena un Array de img, definidos mediante el método createElement de React
- El objeto listaOpciones el cual almacena un Array de <li> y <a> mediante el método createElement de React

```

JS Footer.js X JS index.js
src > JS Footer.js > ...
1 import React,{Component} from 'react';
2
3 const imgrs=new Array("/img/face.jpg","/img/twitter.jpg","/img/instagram.jpg");
4
5 > const listadoOpciones = [ ...
6   ];
7
8   const rs=imgrs.map((origen) =>React.createElement('img', {className:'img-rs', src:origen} ));
9
10  const lista=listadoOpciones.map((it) => React.createElement('li',{className:'li-footer'}, ...
11    | | | React.createElement('a',{href:it.route},it.text)));
12
13  class Footer extends Component{
14
15  }
16
17  export default Footer;
18
19
20
21
22
23
24
25
26

```

Figura 250: Tradiciones.js  
Fuente.- Elaboración Propia

En la clase, al enviar los elementos definidos los agrupo a cada uno a través de un bloque <div>

```

JS Footer.js X JS index.js
src > JS Footer.js > default
20
21  class Footer extends Component{
22    render(){
23      return(
24        <div>
25          <div className="div-footer">
26            {lista}
27          </div>
28          <div className="div-footer">
29            <h3>Síguenos en...</h3>
30            {rs}
31          </div>
32        </div>
33      )
34    };
35  }
36
37
38  export default Footer;

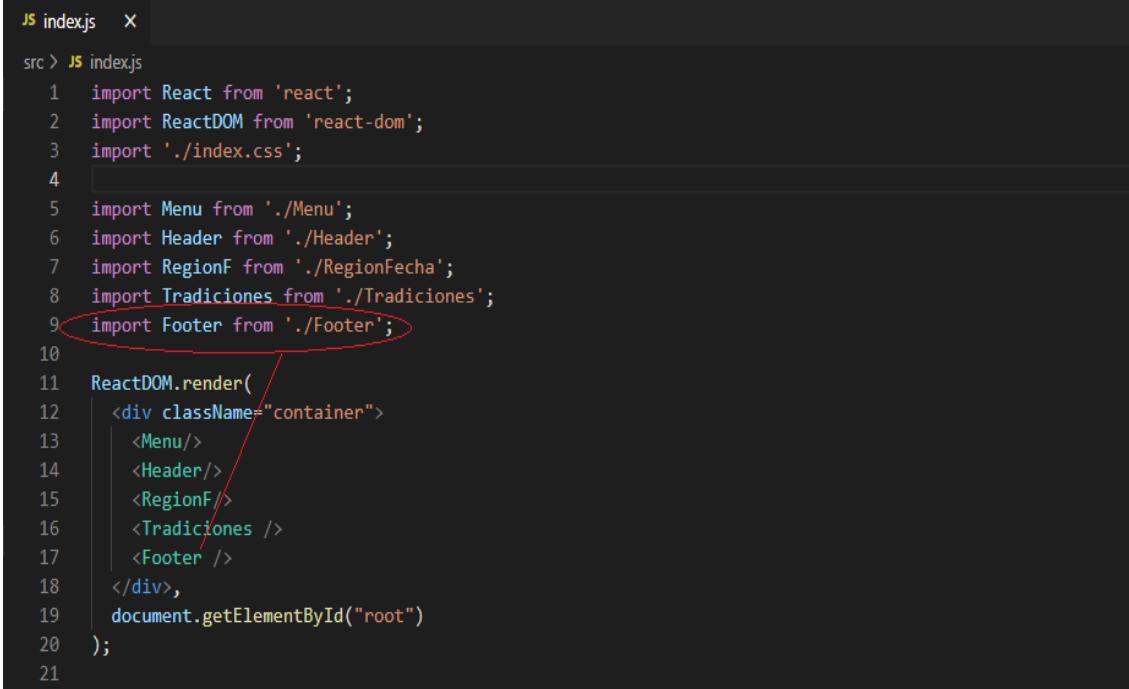
```

Figura 251: Tradiciones.js  
Fuente.- Elaboración Propia

## Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar los archivos Menu.js, Header.js, RegionFecha.js, Tradiciones.js y Footer.js
- Al renderizar la página importamos el contenido de Menu, Header, RegionF, Tradiciones y Footer, tal como se muestra



```

JS index.js ×
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4
5  import Menu from './Menu';
6  import Header from './Header';
7  import RegionF from './RegionFecha';
8  import Tradiciones from './Tradiciones';
9  import Footer from './Footer'; (highlighted line)
10
11 ReactDOM.render(
12   <div className="container">
13     <Menu/>
14     <Header/>
15     <RegionF/>
16     <Tradiciones />
17     <Footer />
18   </div>,
19   document.getElementById("root")
20 );
21

```

Figura 252: index.js  
Fuente.- Elaboración Propia

Actualizar la página para visualizar el contenido de la class Tradición.

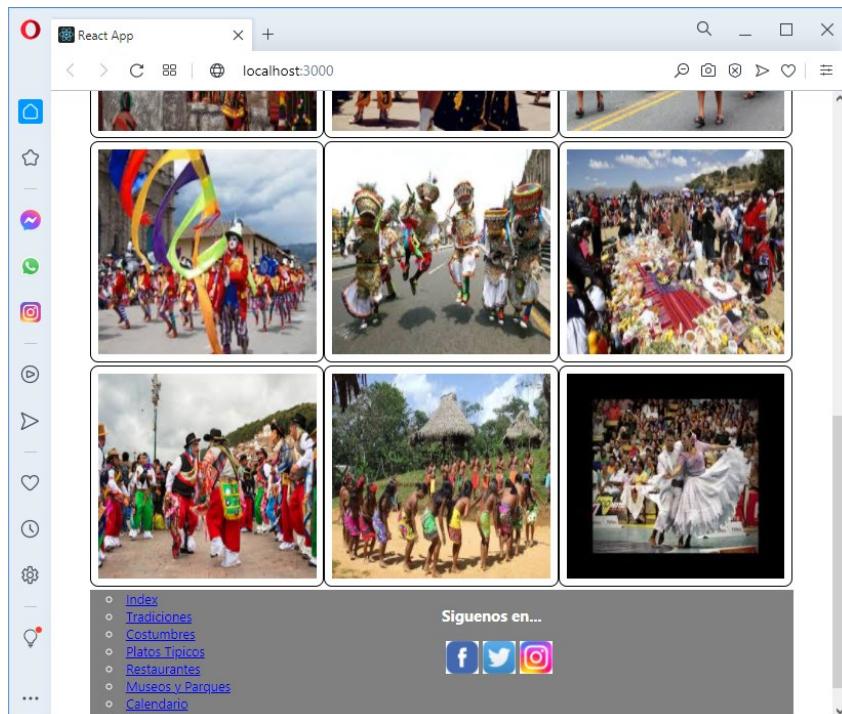


Figura 253 : index.html  
Fuente.- Elaboración Propia

# Resumen

1. JSX es una extensión de la sintaxis de JavaScript que permite describir cómo debería ser la interfaz de usuario. JSX produce “elementos” de React, los cuales no son requeridos, pero son útiles como ayuda visual cuando trabajas con interfaz de usuario en JavaScript.
2. Los componentes funcionales son componentes que generan elementos React. Por convenio se pone el nombre de la función en mayúsculas. Para renderizarlo simplemente se pone una etiqueta con el nombre de la función.
3. Para crear un elemento, importamos los scripts de React y React-Dom, con ellos tenemos acceso a la API de react. React.createElement recibe 3 parámetros básicos:
  - El tipo de elemento html
  - Las propiedades del elemento
  - El contenido del elementoUn elemento creado con react es un object con propiedades, que luego ReactDOM sabe renderizar en el DOM.
4. Un componente en React es una pieza de UI (user interface), que tiene una funcionalidad independiente definida. Un componente se define usando clases, donde una de las formas es utilizar ES6 o ECMAScript 6, la cual nos permite utilizar programación orientada a objetos (POO) para definir un componente.
5. Uno de los recursos elementales que tenemos al desarrollar componentes React son las propiedades. Las propiedades sirven para cargar de datos a los componentes, de modo que éstos puedan personalizar su comportamiento, o para transferir datos de unos componentes a otros para producirse la interoperabilidad entre componentes.
6. El estado de un componente es la memoria en cada momento que tiene la instancia de un componente que se está mostrando en pantalla. Se trata de un atributo de la instancia parecido a las props, al que podremos acceder con this.state.
7. Para modificar el estado de un componente no debemos utilizarlo con la propiedad this.state, sino que tenemos que hacerlo a través de this.setState().

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://es.reactjs.org/docs/getting-started.html>
- <https://ichi.pro/es/uso-de-temporizadores-en-aplicaciones-react-182487945986495>
- <https://stackoverflow.com/questions/36270422/reactjs-settimeout-not-working/36270461>
- <https://reactgo.com/settimeout-in-react-hooks/>
- <https://desarrolloweb.com/articulos/estado-componentes-react.html>
- <https://es.reactjs.org/docs/state-and-lifecycle.html>
- <https://desarrolloweb.com/articulos/propiedades-estado-predeterminado-componentes-react.html>

## 6.3. CICLO DE VIDA Y EVENTOS EN REACT

### 6.3.1. Ciclo de vida de los componentes

El ciclo de vida no es más que una serie de estados por los cuales pasa todo componente a lo largo de su existencia. Esos estados tienen correspondencia en diversos métodos, que podemos implementar para realizar acciones cuando se van produciendo.

En React es fundamental el ciclo de vida, porque hay determinadas acciones que debemos necesariamente realizar en el momento correcto de ese ciclo.

Un ciclo de vida de un componente está clasificado en tres estados:

- El montaje se produce la primera vez que un componente va a generarse, incluyéndose en el DOM.
- La actualización se produce cuando el componente ya generado se está actualizando.
- El desmontaje se produce cuando el componente se elimina del DOM.

Además, dependiendo del estado actual de un componente y lo que está ocurriendo con él, se producirán grupos diferentes de etapas del ciclo de vida. En la siguiente imagen puedes ver un resumen de esta diferenciación. Observarás que en el primer renderizado de un componente se pasa por etapas del ciclo de vida diferentes de las que se pasa cuando se produce un cambio en sus propiedades o un cambio en el estado, o su desmontaje.

Primer renderizado	Cambios en las propiedades	Cambio en el estado	Componente se desmonta
get defaultProps *	componentWillReceiveProps	shouldComponentUpdate	componentWillUnmount
getInitialState *	shouldComponentUpdate	componentWillUpdate	
componentWillMount	componentWillUpdate	render *	
render *	render *	componentDidUpdate	
componentDidMount	componentDidUpdate		

Figura 254 : Ciclo de vida

Fuente. – Tomado de <https://desarrolloweb.com/articulos/ciclo-vida-componentes-react.html>

### Descripción de los métodos del ciclo de vida

Ahora vamos a describir las distintas etapas del ciclo de vida de los componentes .

Etapa del ciclo de vida	Descripción
componentWillMount()	Método de montaje. Se ejecuta justo antes del primer renderizado del componente. Si dentro de este método seteas el estado del componente con setState(), el primer renderizado mostrará ya el dato actualizado y sólo se renderizará una vez el componente.
componentDidMount()	Método de montaje, se ejecuta en el lado del cliente. Se produce inmediatamente después del primer renderizado. Una vez se invoca este método ya están

	disponibles los elementos asociados al componente en el DOM y sus componentes hijos. Sitio para realizar llamadas Ajax, setIntervals, etc.
componentWillReceiveProps(nextProps)	Método de actualización que se invoca cuando las propiedades se van a actualizar. Recibe el valor futuro del objeto de propiedades que se va a tener. El valor anterior es el que está todavía en el componente, pues este método se invocará antes de que esos cambios se hayan producido.
shouldComponentUpdate(nextProps, nextState)	Método de actualización el cual retorna un valor booleano. Si devuelve <b>verdadero</b> el componente se debe renderizar de nuevo y si recibe <b>falso</b> quiere decir que el componente no se debe de renderizar de nuevo. Se invocará tanto cuando se producen cambios de propiedades o cambios de estado y es una oportunidad de comprobar si esos cambios debiesen producir una actualización en la vista del componente. Por defecto (si no se definen) devuelve siempre true, para que los componentes se actualicen ante cualquier cambio de propiedades o estado. El motivo de su existencia es la optimización, puesto que el proceso de renderizado tiene un coste y podemos evitarlo si realmente no es necesario de realizar. Este método recibe dos parámetros con el conjunto de propiedades y estado futuro.
componentWillUpdate(nextProps, nextState)	Método de actualización se invocará justo antes de que el componente vaya a actualizar su vista. Es indicado para tareas de preparación de esa inminente renderización causada por una actualización de propiedades o estado.
componentDidUpdate(prevProps, prevState)	Método de actualización que se ejecuta después de actualizar un componente. En este paso los cambios ya están trasladados al DOM del navegador, así que podríamos operar con el DOM para hacer nuevos cambios. Como parámetros en este caso recibes el valor anterior de las propiedades y el estado.
componentWillUnmount()	Este es el único método de desmontado y se ejecuta en el momento que el componente se va a retirar del DOM. Este método es muy importante, porque es el

	momento en el que se debe realizar una limpieza de cualquier cosa que tuviese el componente y que no deba seguir existiendo cuando se retire de la página. Por ejemplo, temporizadores o manejadores de eventos que se hayan generado sobre partes del navegador que no dependen de este componente.
--	--

En este ejemplo, vemos como el método componentDidMount(), ejecuta el método setInterval() invocando al método \_tick() por cada segundo una vez que los elementos del DOM (<h2>) se encuentra disponible, y el método componentWillUnmount() donde desactiva el setInterval: clearInterval.

```

JS index.js JS Reloj.js X Settings index.html
src > JS Reloj.js > Reloj > componentDidMount
4
5   class Reloj extends Component{
6     constructor(props) {
7       super(props);
8       this.state = {
9         tiempo: f.getHours()+":"+f.getMinutes()+":"+f.getSeconds()
10      };
11    }
12
13    componentWillUnmount() {
14      clearInterval(this.interval);
15    }
16
17    componentDidMount() {
18      this.interval = setInterval(() => { this._tick(); }, 1000);
19    }
20
21    _tick() {
22      f=new Date();
23      this.setState((state, props) => ({
24        tiempo: f.getHours()+":"+f.getMinutes()+":"+f.getSeconds(),
25      }));
26    }
27    render(){
28      return <h2>Son las {this.state.tiempo} </h2>
29    };
30  }
31

```

Figura 255 : Ciclo de vida - ejemplo

Fuente.- Elaboración Propia

### 6.3.2. Eventos en react

Los eventos en React se definen generalmente de manera declarativa, en el código de la vista o template, producido con JSX en el método render().

React tiene un sistema de eventos sintéticos que ejecutan una acción cuando ocurre un acontecimiento. Por ejemplo, cuando haga clic en un botón dentro de un componente que hemos definido.

Las declaraciones de eventos en React tienen siempre esa forma: el prefijo "on", seguido del tipo de evento que queremos capturar (onClick, onInput...). Esta definición se llama **camelCase**: onClick, onLoad, onScroll, etc.

Vamos a ver un ejemplo. Queremos escuchar un evento de click desde un botón que declaramos con JSX. Escribiremos el botón (`<button>texto</button>`) y en un atributo `onClick` (ojo con la mayúscula) añadiremos la función "Mensaje", que será la reacción.

Quedará así:

```

3  class Eventos extends Component{
4
5      Mensaje(){
6          alert("Bienvenido");
7      };
8
9      render(){
10         return <button onClick={this.Mensaje}>Hazme Click</button>
11     };
12 }

```

Figura 256 : Eventos  
Fuente.- Elaboración Propia

Podríamos escribir directamente Mensaje como una arrow function y ejecutarla en el evento del elemento. Cuando hagamos clic en el botón, React se encargará de escuchar el evento y de ejecutar la función.

```

3  class Eventos extends Component{
4
5      Mensaje = (event) => {
6          alert("Bienvenido");
7      };
8
9      render(){
10         return <button onClick={this.Mensaje}>Hazme Click</button>
11     };
12 };
13

```

Figura 257: Eventos  
Fuente.- Elaboración Propia

## Eventos más usados

### Escuchadores de eventos de ratón

<code>onClick</code>	botón izquierdo del ratón
<code>onMouseOver</code>	pasar el ratón sobre un elemento
<code>onMouseOut</code>	sacar el ratón del elemento

### Escuchadores de eventos de teclado

<code>onKeyPress</code>	pulsar una tecla
-------------------------	------------------

### Escuchadores de eventos sobre elementos

<code>onFocus</code>	poner el foco (seleccionar) en un elemento, por ejemplo un <code>&lt;input&gt;</code>
----------------------	---

onChange	al cambiar el contenido de un <input> , <textarea> o <select> (no es necesario quitar el foco del input para que se considere un cambio, al contrario que en el DOM)
----------	--

### Escuchadores de eventos de formularios

onSubmit	pulsar el botón submit del formulario
onReset	pulsar el botón reset del formulario

### PreventDefault

Para prevenir el comportamiento por defecto de un evento, debemos ejecutar preventDefault() en vez de retornar false.

Por ejemplo prevenir que un <a> abra una nueva página. En la función Mensaje(e), primero ejecuta e.preventDefault() para que no ejecute su evento, para luego ejecutar un alert().

```

3 function Mensaje(e) {
4     e.preventDefault();
5     alert("Bienvenido");
6 };
7 class Eventos extends Component{
8
9     render(){
10         return <a href="#" onClick={Mensaje}>Saludo</a>
11     };
12 };

```

Figura 258: PreventDefault

Fuente.- Elaboración Propia

### Bindear el contexto para acceder a this

Al implementar un manejador de evento es habitual que queramos acceder a las propiedades o métodos del propio componente. Por ejemplo, en el método Marcar() queremos cambiar el estado de los componentes y por tanto necesitamos manipularlo con this.setState().

Sin embargo aquí nos encontramos con una dificultad que viene del propio lenguaje JavaScript no de React específicamente. En las funciones no es posible acceder a this como referencia al objeto sobre el que se invoca el método. Es por ello por lo que necesitamos **bindear** el contexto.

Si no enlazamos *this.Marcar* y lo pasáramos a onClick, this sería undefined cuando se llame la función. Es muy importante hacer el binding.

```

3  const like='/iconos/like.jpg';
4  const dislike='/iconos/dislike.jpg';
5
6  class Eventos extends Component{
7      constructor(props){
8          super(props);
9          this.state={
10              marca:false,
11              origen:like
12          }
13          this.Marcar = this.Marcar.bind(this)
14      };
15
16      Marcar = (event) => {
17          this.setState({
18              marca:this.state.marca==true ? false : true,
19              origen:this.state.marca==true ? like : dislike
20          });
21      };
22
23      render(){
24          return <a href="#" onClick={this.Marcar}><img src={this.state.origen}></a>
25      };
26  };
27

```

Figura 259 : bind()  
Fuente.- Elaboración Propia

### Pasando parámetros al controlador de evento o callback

Para que exista comunicación entre los componentes, debemos definir controladores o callback, donde al pasar valores o parámetros, los valores de los estados podrán ser actualizados por dichos parámetros.

En este ejercicio definimos el controlador Clickear con un parámetro id, el cual, al momento de ejecutar dicho controlador (tal como se ve en la figura), enviamos un valor donde el parámetro id lo recibe para efectuar un cálculo.

```

1  import React,{Component} from 'react';
2
3  class Eventos2 extends Component{
4
5      Clickear(id) {
6          return event => { alert("Click en el elemento:"+id); }
7      };
8
9      render(){
10         return(
11             <div>
12                 <ul>
13                     <li onClick={this.Clickear(0)}><a href="#">Primer Elemento</a></li>
14                     <li onClick={this.Clickear(1)}><a href="#">Segundo Elemento</a></li>
15                     <li onClick={this.Clickear(2)}><a href="#">Tercer Elemento</a></li>
16                 </ul>
17             </div>
18         );
19     };
20 }
21

```

Figura 260 : Parámetros  
Fuente.- Elaboración Propia

En este ejercicio hemos definimos en las etiquetas HTML el atributo data-op el cual cada uno almacena un valor. Definimos el controlador Clickear con un parámetro e, el cual, lee el valor del atributo data-op: e.target.dataset.op para ser visualizados.

```

1 import React,{Component} from 'react';
2
3 class Eventos3 extends Component{
4
5     Clickear =(e,event) => {
6         alert("Click en el elemento:"+e.target.dataset.op)
7     };
8
9     render(){
10         return(
11             <div>
12                 <ul>
13                     <li ><a href="#" data-op={0} onClick={this.Clickear}>Primer Elemento</a></li>
14                     <li ><a href="#" data-op={1} onClick={this.Clickear}>Segundo Elemento</a></li>
15                     <li ><a href="#" data-op={2} onClick={this.Clickear}>Tercer Elemento</a></li>
16                 </ul>
17             </div>
18         );
19     }
20 }
21

```

Figura 261 : parámetros y el atributo data  
Fuente.- Elaboración Propia

### 6.3.3. Condicionales en templates JSX de react

El renderizado condicional en React funciona de la misma forma que lo hacen las condiciones en JavaScript. Usa operadores de JavaScript como if o el operador condicional para crear elementos representando el estado actual, y deja que React actualice la interfaz de usuario para emparejarlos.

Para crear un condicional en un template de React podemos usar una expresión con el mismo código JavaScript que usaríamos en la programación tradicional, aplicando el operador ternario.

Como sabemos ya, las expresiones en JSX se escriben entre llaves. Ahí podemos colocar código JavaScript y aquel valor que resulte de la evaluación de la expresión es lo que aparecerá como contenido en el template.

La expresión condicional nos quedará de manera similar a esta:

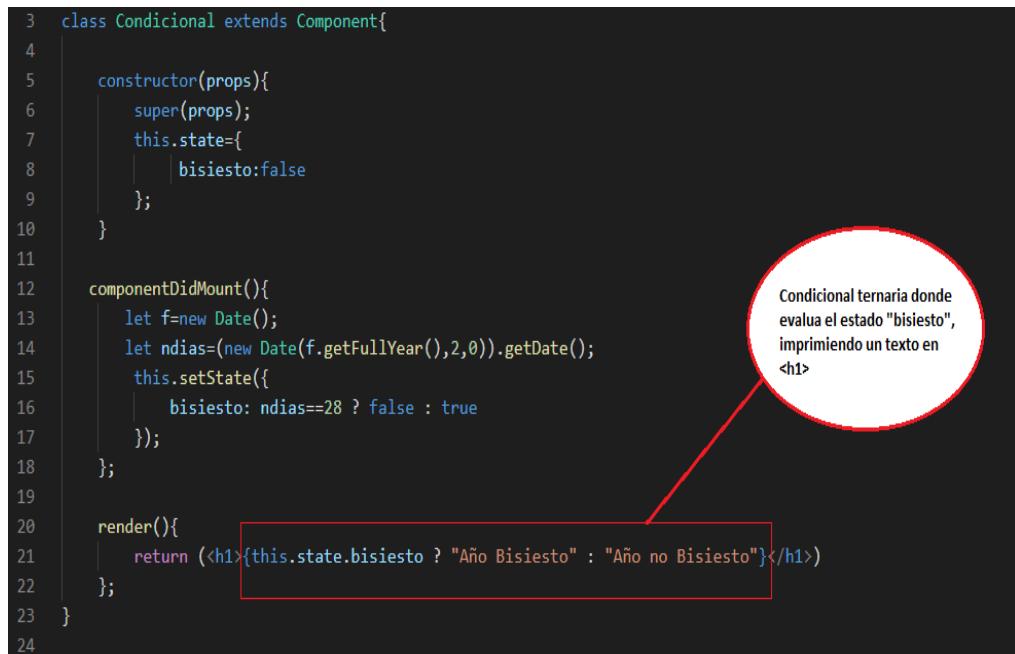
```
return (<h1>{this.state.bisiesto ? "Año Bisiesto" : "Año no Bisiesto"}</h1>)
```

Figura 262 : condicionales en templates  
Fuente.- Elaboración Propia

Para el procesamiento del condicional se evalúa un atributo del estado "bisiesto". En caso positivo se muestra en un heading H1 la expresión “Año Bisiesto”. En caso contrario, simplemente mostramos la expresión “Año no Bisiesto”

En este caso hemos usado poco código JSX para cada parte del condicional, pero podríamos tener unas cuantas etiquetas o componentes que mostrar para cada caso.

No es un problema, puesto que podemos seguir usando la misma expresión con el operador ternario. Sólo entonces convendría organizar el código de una manera que resultase fácil para la lectura. Por ejemplo tal como se puede ver a continuación.



```

3  class Condicional extends Component{
4
5    constructor(props){
6      super(props);
7      this.state={
8        |   bisiesto:false
9      };
10     }
11
12    componentDidMount(){
13      let f=new Date();
14      let ndias=(new Date(f.getFullYear(),2,0)).getDate();
15      this.setState({
16        |   bisiesto: ndias==28 ? false : true
17      });
18    }
19
20    render(){
21      return (<h1>{this.state.bisiesto ? "Año Bisiesto" : "Año no Bisiesto"}</h1>)
22    };
23  }
24

```

A red callout points to the conditional ternary expression in the render() method:

Condicional ternaria donde evalua el estado "bisiesto", imprimiendo un texto en <h1>

Figura 263: condicionales en templates

Fuente.- Elaboración Propia

#### 6.3.4 Creación de repeticiones en templates JSX con react

Para la creación elementos HTML se realiza mayormente usando una herramienta del propio lenguaje JavaScript, el método **map()** disponible en los Arrays.

Este método sirve para producir un nuevo array a partir de una repetición por cada uno de los elementos del array inicial.



```

2
3  let lista=new Array("Marca","Precio","Mas comprados","Talla","Todas");
4
5  class Repeticiones extends Component{
6
7    render(){
8      let items=lista.map((x) => <li>{x}</li>);
9      return (
10        <div>
11          <h3>Filtrar por</h3>
12          {items}
13        </div>
14      );
15    }
16  }
17
18  export default Repeticiones;
19

```

A red callout points to the line where the map() method is used:

A partir del Array lista, definimos un Array llamado items, que almacena elementos <li>

Figura 264: Creación de repeticiones

Fuente.- Elaboración Propia

### Usar llaves (keys) para los ítems de la repetición

Si ordenamos el array dinámicamente de distintos modos los índices cambiarán y el componente podría comenzar a funcionar de manera errática. Incluso, aún en el caso que no se necesite ordenar el array dinámicamente, podemos encontrarnos ante varias desventajas, como una caída del rendimiento.

Lo más normal es que los datos a listar te lleguen desde una base de datos. En este caso, podrías usar las propias claves primarias de los elementos como índices en el listado.

```

3 let lista= [{id:1,Texto:"Marca"},  

4   |   | {id:2,Texto:"Precio"},  

5   |   | {id:3,Texto="Mas comprados"},  

6   |   | {id:4,Texto:"Talla"},  

7   |   | {id:5,Texto:"Todas"}];  

8  

9 class Repeticiones2 extends Component{  

10  

11   render(){  

12     let items=lista.map((x) => <li key={x.id}>{x.Texto}</li>);  

13     return (  

14       <div>  

15         <h3>Filtrar por</h3>  

16         {items}  

17       </div>  

18     )  

19   };  

20 }  

21

```

Figura 265: Creación de repeticiones con key

Fuente: Diseño propio

### 6.3.5 Extendiendo componentes con mixins

La idea bajo React es crear componentes reusables. Si varios de nuestros componentes poseen la misma funcionalidad, podemos crear un módulo y reusarlo en todos ellos. Los **mixins** nos permiten hacer eso.

#### Definir un mixin

Para crear un mixin hay que crear un objeto con los métodos necesarios para agregar la funcionalidad. Podemos usar los métodos de los componentes de React (como componentWillMount o componentWillUnmount), con la ventaja de que, si usamos varios mixins que definen los mismos métodos, se ejecutarán ambos, uno detrás de otro.

## LABORATORIO 1

### Implementando eventos MouseOver y MouseOut en una aplicación React JS

En este laboratorio vamos a implementar los eventos MouseOver y MouseOut a las imágenes de las redes sociales, donde al pasar el mouse cambiamos la imagen y al salir el mouse visualizamos la imagen original.

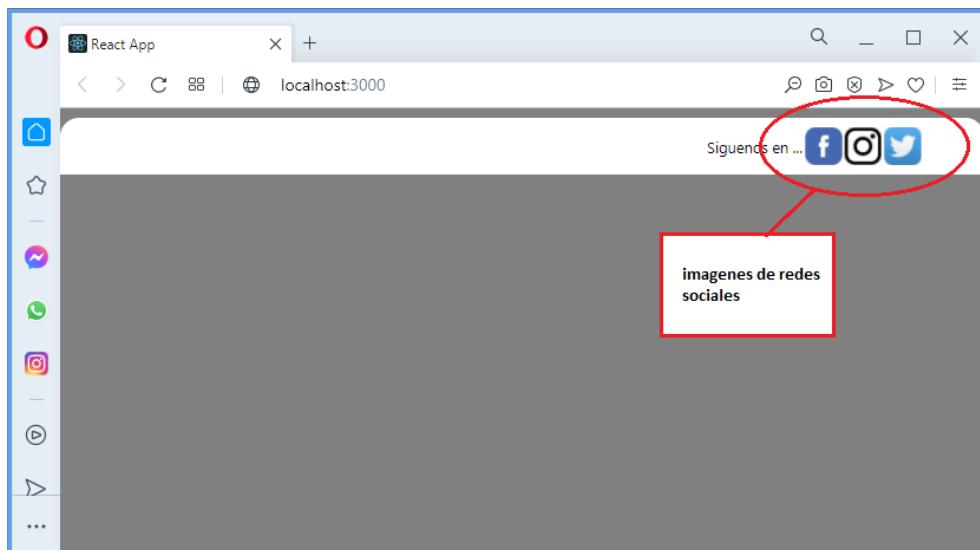


Figura 266: Diseño del bloque de redes sociales  
Fuente.- Elaboración Propia

### Agregando selectores a index.css

Primero agregamos en el archivo index.css, los selectores de class y definimos sus propiedades.

```
JS Header_social.js  # index.css X
src > # index.css > span
15
16 .divsocial{
17   width: 25%;
18   height: auto;
19   float: left;
20   margin-top: 10px;
21   text-align: right;
22   padding-right: 5%;
23   padding-left: 70%;
24   background-color: white;
25   border-radius: 20px 20px 0 0;
26 }
27 .rsocial{
28   width: 40px;
29   height: 40px;
30   margin-top: 10px;
31   margin-bottom: 10px;
32   margin-left: 2px;
33   float: left;
34 }
35
36 span{
37   margin-top:20px;
38   float: left;
39 }
```

definir el selector de class: divsocial

definir el selector de class: rsocial, para imágenes de redes sociales

definir el selector span

Figura 267 : index.css  
Fuente.- Elaboración Propia

### Trabajando con Header\_social.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Header\_social.js.

- En el archivo importar React y {Component}
- Defina los Array rs y rschange, el cual almacena las direcciones de imágenes a utilizar en los eventos mouseOver y mouseOut
- A continuación, defina la clase Header\_social de tipo Component

```

JS Header_social.js X
src > JS Header_social.js > ...
1 import React,{Component} from 'react';
2
3 const rs=new Array("/img/social/face.jpg","/img/social/instagram.jpg",
4 | | | | "/img/social/twitter.jpg");
5
6 const rschange=new Array("/img/social/face1.jpg","/img/social/instagram1.jpg",
7 | | | | "/img/social/twitter1.jpg");
8
9 class Header_social extends Component{
10
11 }
12
13 export default Header_social;
14

```

definir los Arrays, rs y rschange para los cambios de imágenes

Defina la clase Header\_social de tipo Component

Figura 268: Header\_social.js  
Fuente.- Elaboración Propia

En la clase Header\_social:

- En el render, defina el Array imágenes, el cual almacena una lista de elementos <img> cada uno con su atributo src y sus eventos onMouseOver y onMouseOut, programando cada evento.

```

JS Header_social.js X
src > JS Header_social.js > ...
8 class Header_social extends Component{
9
10 render(){
11   let imagenes = rs.map((x,index) =>
12     <img src={x} className="rsocial"
13       onMouseOver={ e => e.currentTarget.src=rschange[index] }
14       onMouseOut={e => e.currentTarget.src=rs[index]} />
15 );
16
17   return(
18     <div className="divsocial">
19       <span>Síguenos en ...</span>
20       {imagenes}
21     </div>
22   );
23 }
24
25 export default Header_social;
26

```

utilizando el metodo map, definimos un Array de img, cada uno con su src y sus eventos

renderizamos, imágenes a la class

Figura 269: Header\_social.js  
Fuente.- Elaboración Propia

## Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar el archivo Header\_socil.js y lo llamaremos Social
- Al renderizar la página importamos el contenido de Social



```
JS index.js X
src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import Social from './Header_social';
5
6 ReactDOM.render(
7   <Social />,
8   document.getElementById('root')
9 );
10
```

Figura 270: Index.js  
Fuente.- Elaboración Propia

En la ventana de consola, ejecutar el proyecto React JS: **npm start sesion03react** y dar Enter, donde se visualiza el contenido de Header\_social.js

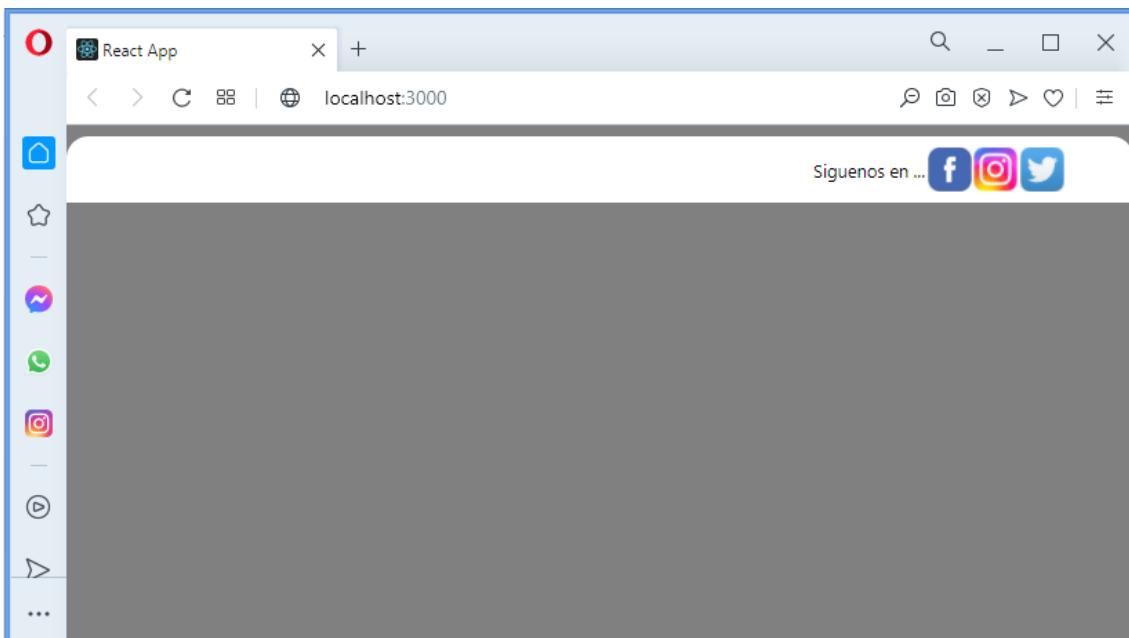


Figura 271: Index.html  
Fuente.- Elaboración Propia

## LABORATORIO 2

### Implementando un carrusel de imágenes en una aplicación React JS

En este laboratorio vamos a implementar un carrusel de imágenes, donde las imágenes se cambian por intervalo de un segundo. Además, este proceso tiene un numeral donde al hacer click, cambia la imagen de la cabecera y continua el carrusel.

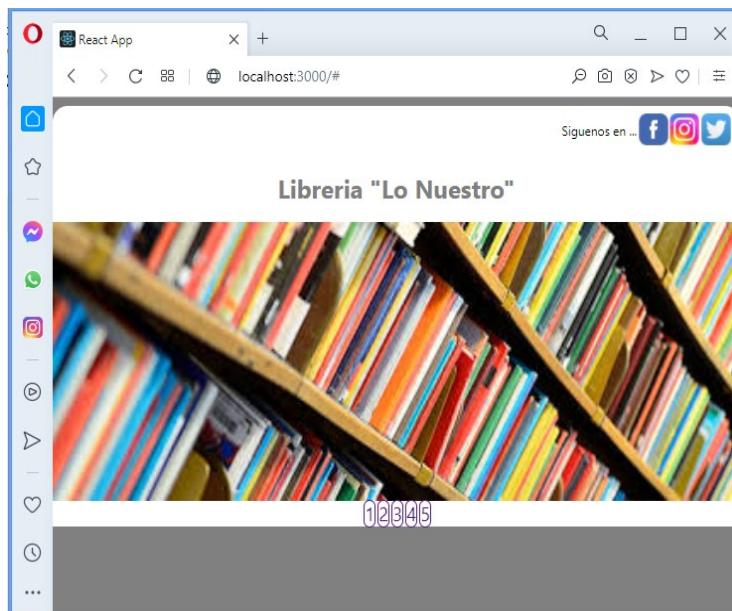


Figura 272: Diseño de la cabecera

Fuente.- Elaboración Propia

### Agregando selectores a index.css

Primero agregamos en el archivo index.css, los selectores de class y definimos sus propiedades.

```
# index.css x JS index.js JS Header.js JS Eventos4.js
src > # index.css ...
41 .divheader{
42   width: 100%;
43   float: left;
44   background-color: white;
45 }
46
47 .h1cab{
48   color: gray;
49   text-align: center;
50 }
51 .imgcab{
52   width: 100%;
53   height: 350px;
54   float: left;
55 }
56
57 .divitem{
58   text-align: center;
59 }
60
61 .item{
62   font-size: 1.5em;
63   margin-left: 3px;
64   border: 1px solid;
65   text-decoration: none;
66   border-radius: 10px;
67 }
```

definir el selector de class divheader, estructura bloque para los elementos de la cabecera

definir el selector de class, h1cab (título)

definir el selector de class, imgcab (imagen de la cabecera)

definir el selector de class divitem (bloque para los numeros del header)

definir el selector de class, item (los numeros del header)

Figura 273: selectores del index.css

Fuente.- Elaboración Propia

### Trabajando con Header.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Header.js.

- En el archivo importar React y {Component}
- Defina el Array imágenes, el cual almacena las direcciones de imágenes a utilizar en el carrusel de imágenes
- A continuación, defina la clase Header de tipo Component

```

JS index.js JS Header.js X
src > JS Header.js > ...
1 import React,{Component} from 'react';
2
3 const imagenes=["/img/header/img1.jpg","/img/header/img2.jpg","/img/header/img3.jpg",
4 "/img/header/img4.jpg","/img/header/img5.jpg"];
5
6 class Header extends Component{
7
8 };
9
10 export default Header;
11

```

defina una Array de imágenes

defina la clase Header de tipo Component

Figura 274: Header.js

Fuente.- Elaboración Propia

Para iniciar, definimos el constructor de la clase, el cual inicializa dos estados: rotación y el estado "c" (el cual almacena la ubicación de la imagen a visualizar)

```

JS index.js JS Header.js X
src > JS Header.js > Header
1 import React,{Component} from 'react';
2
3 const imagenes=["/img/header/img1.jpg","/img/header/img2.jpg","/img/header/img3.jpg",
4 "/img/header/img4.jpg","/img/header/img5.jpg"];
5
6 class Header extends Component{
7   constructor(props){
8     super(props);
9     this.state={
10       rotacion:null,
11       c:0
12     };
13   }
14 };
15

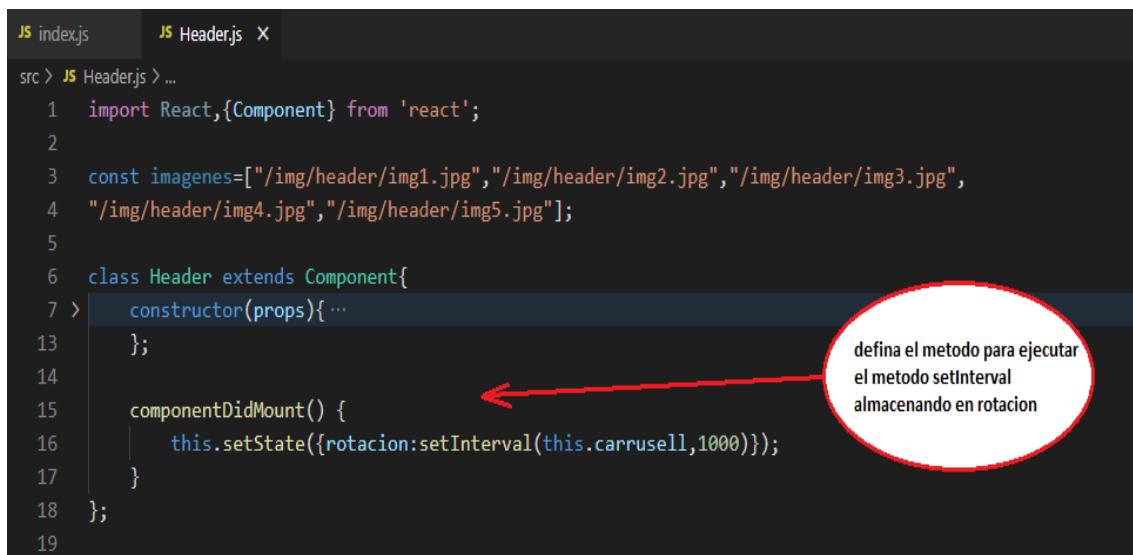
```

inicialice en el constructor de la clase los estados: rotacion y c

Figura 275: Header.js

Fuente.- Elaboración Propia

A continuación, definimos al método componentDidMount(), una vez que los estados y elementos HTML estén definidos, el método actualiza el valor de rotación ejecutando en el setInterval, el método carrusel, por intervalo de 1 segundo.



```

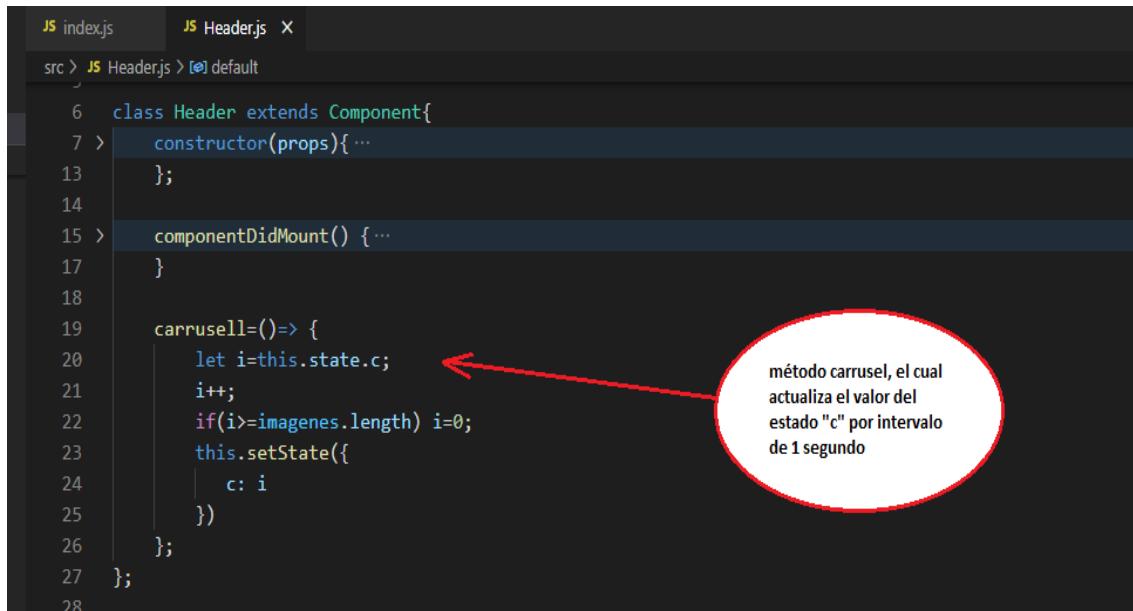
JS index.js      JS Header.js X
src > JS Header.js > ...
1 import React,{Component} from 'react';
2
3 const imagenes=["/img/header/img1.jpg","/img/header/img2.jpg","/img/header/img3.jpg",
4 "/img/header/img4.jpg","/img/header/img5.jpg"];
5
6 class Header extends Component{
7 >   constructor(props){ ...
8     }
9
10    componentDidMount() {
11      this.setState({rotacion:setInterval(this.carrusell,1000)});
12    }
13  };
14
15 };
16
17
18
19

```

defina el metodo para ejecutar el metodo setInterval almacenando en rotacion

Figura 276: Header.js  
Fuente.- Elaboración Propia

Definido el setInterval, el cual ejecuta el método carrusel() por intervalo de 1 segundo, defina el método carrusel() el cual actualiza el valor del estado “c”, incrementándolo de 1 en 1.



```

JS index.js      JS Header.js X
src > JS Header.js > [e] default
1
2
3
4
5
6 class Header extends Component{
7 >   constructor(props){ ...
8     }
9
10    componentDidMount() { ...
11    }
12
13
14
15    carrusell=()=> {
16      let i=this.state.c;
17      i++;
18      if(i>=imagenes.length) i=0;
19      this.setState({
20        c: i
21      });
22    };
23
24
25
26
27
28

```

método carrusel, el cual actualiza el valor del estado "c" por intervalo de 1 segundo

Figura 277: Header.js  
Fuente.- Elaboración Propia

Luego, defina el evento Numeral, el cual realiza el cambio de imagen del carrusel actualizando el valor del estado "c", para ello primero detiene la rotación de imágenes y luego la reinicia.

```

JS index.js JS Header.js X
src > JS Header.js > default
      ...
6   class Header extends Component{
7     constructor(props){ ...
8   }
9
10  componentDidMount() { ...
11  }
12
13  carrusell=()=> { ...
14  }
15
16  Numeral = (e,event) => {
17    clearInterval(this.state.rotacion);
18    this.setState({
19      c: e.target.dataset.id
20    })
21    this.setState({rotacion:setInterval(this.carrusell,1000) });
22  }
23
24  ...
25
26  ...
27
28  ...
29
30  ...
31
32  ...
33
34  ...
35
36  ...
37

```

Figura 278: Header.js  
Fuente.- Elaboración Propia

En el render, defina el Array de ítems, el cual almacena una lista de elementos <d> cada uno con su atributo data-id y su evento onClick. Envío los elementos dentro del return

```

JS index.js JS Header.js X
src > JS Header.js > ...
      ...
6   class Header extends Component{
7     constructor(props){ ...
8   }
9
10  componentDidMount() { ...
11  }
12
13  carrusell=()=> { ...
14  }
15
16  Numeral = (e,event) => { ...
17  }
18
19  render(){
20    let items=imagenes.map((x,index) =>
21      <a href="#" className="item" data-id={index} onClick={this.Numeral}>{index+1}</a>
22    )
23
24    return(
25      <div className="divheader">
26        <h1 className="h1cab">Librería "Lo Nuestro"</h1>
27        <img className="imgcab" src={imagenes[this.state.c]} />
28        <div className="divitem">
29          {items}
30        </div>
31      </div>
32    );
33  }
34
35  ...
36
37  ...
38
39  ...
40
41  ...
42
43  ...
44
45  ...
46
47  ...
48
49  ...
50

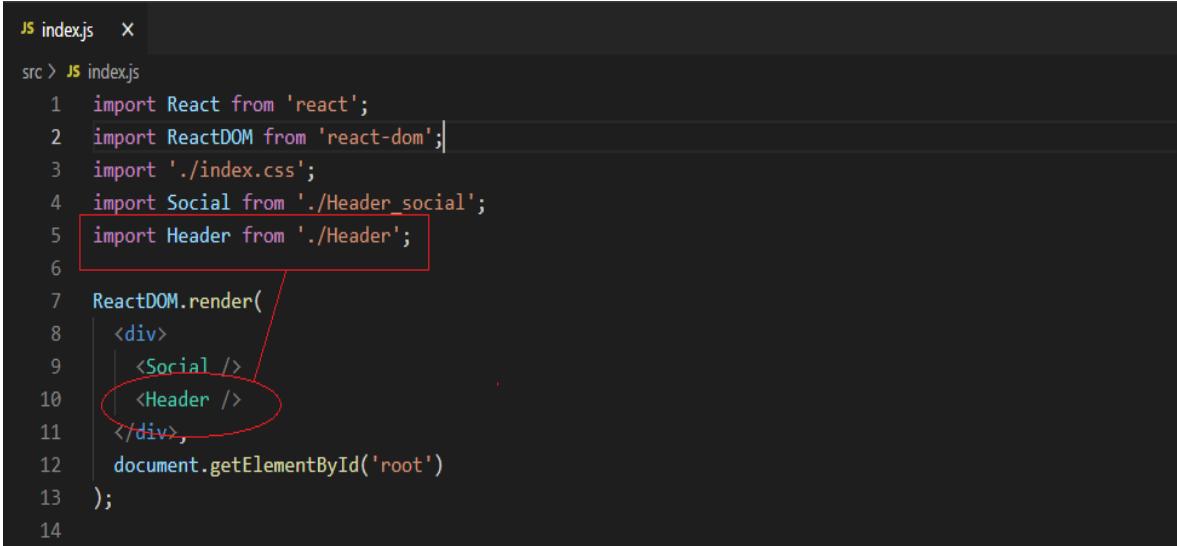
```

Figura 279: Header.js  
Fuente.- Elaboración Propia

### Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar el archivo Header.js y lo llamaremos Header
- Al renderizar la página importamos el contenido de Header



```
JS index.js X
src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import Social from './Header_social';
5 import Header from './Header';
6
7 ReactDOM.render(
8   <div>
9     <Social />
10    <Header />
11   </div>,
12   document.getElementById('root')
13 );
14
```

Figura 280: Index.js  
Fuente.- Elaboración Propia

Actualiza la página index.html, para visualizar el contenido de Header.js

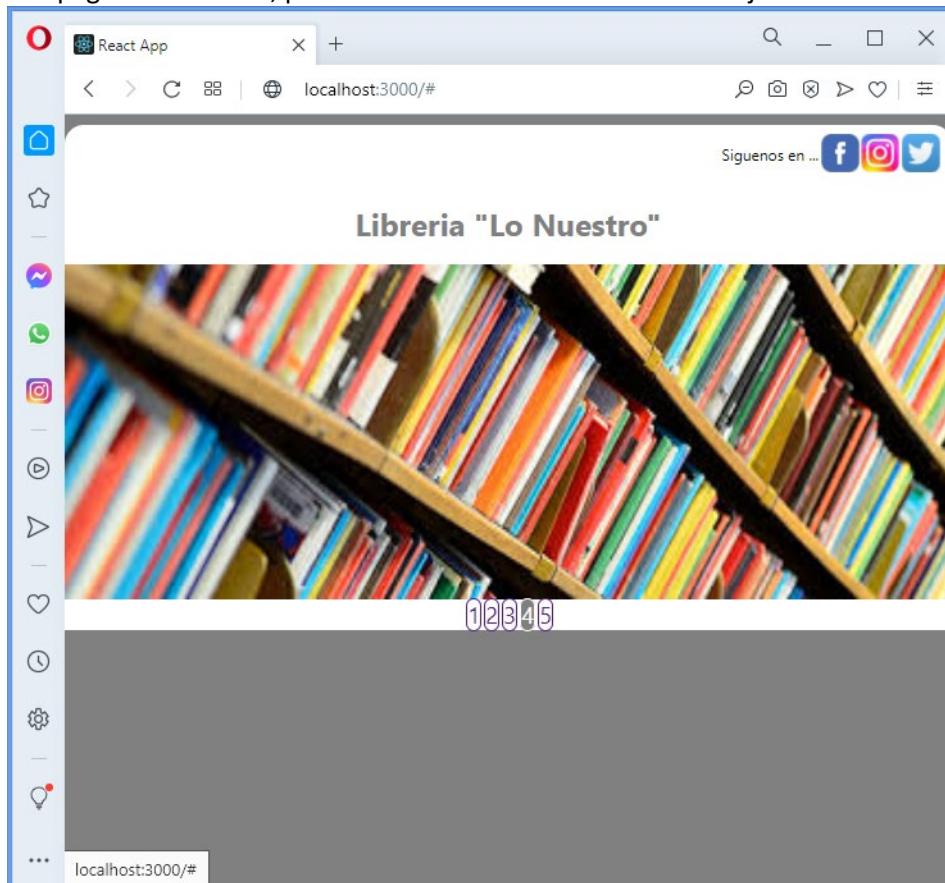


Figura 281: Index.html  
Fuente.- Elaboración Propia

## LABORATORIO 3

### Implementando un filtro para buscar libros en una aplicación React JS

En este laboratorio vamos a implementar una búsqueda de libros, ingresando las iniciales del título del libro por un <input>.

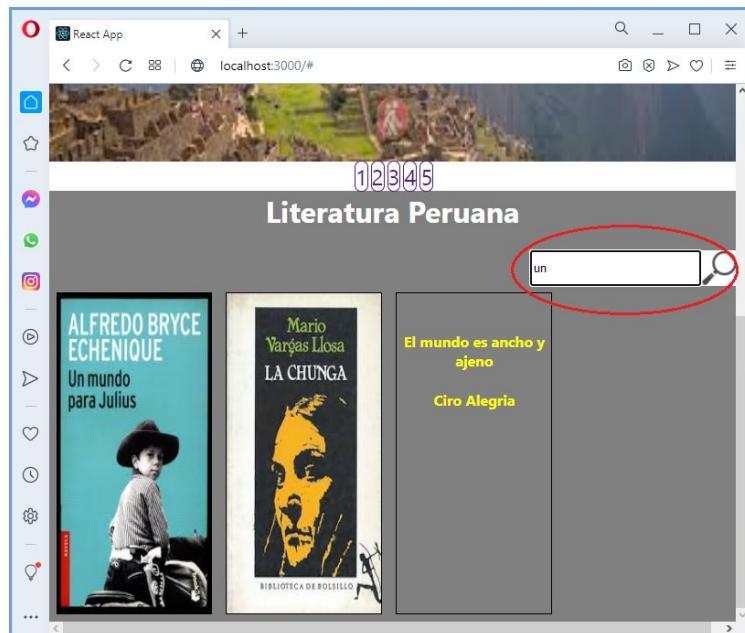


Figura 282: Diseño de búsqueda de libros

Fuente.- Elaboración Propia

### Agregando selectores a index.css

Primero agregamos en el archivo index.css, los selectores de class y definimos sus propiedades

```

JS index.js      JS Portal_Libros.js    # index.css X
src > # index.css > ...
73
74   .art-libro{
75     width: 22.5%;
76     height: 350px;
77     margin:1%;
78     border: 1px solid;
79     float: left;
80     overflow: hidden;
81   }
82
83   .img-libro{
84     width: 100%;
85     height: 350px;
86   }
87
88   .h1-libro{
89     text-align: center;
90     color: black;
91   }
92
93   .h4-libro{
94     text-align: center;
95     color: yellow;
96   }
97
98   .art-libro:hover .img-libro{
99     height: 0px;
100    transition: all 0.5s;
101  }
102

```

definir el selector de class art-libro, para agrupar la imagen, título y autor

definir el selector de class img-libro

definir el selector de class h1-libro (título)

definir el selector de class h4-libro

definir el hover de img-libro

Figura 283: Index.css

Fuente.- Elaboración Propia

```

JS index.js JS Portal_Libros.js # index.css X
src > # index.css > h4-libro
102
103 .div-buscar{
104   width: 100%;
105   height: auto;
106   display: grid;
107   grid-template-columns: 70% 25% 5%;
108   float: left;
109 }
110
111 .img-buscar{
112   width: 40px;
113   float: left;
114 }

```

Figura 284: Index.css

Fuente.- Elaboración Propia

### Trabajando con Portal\_Libros.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Portal\_Libros.js.

- En el archivo importar React y {Component}
- Importamos la colección de libros el cual llamaremos colección
- A continuación, defina la clasePortal\_Libros de tipo Component

```

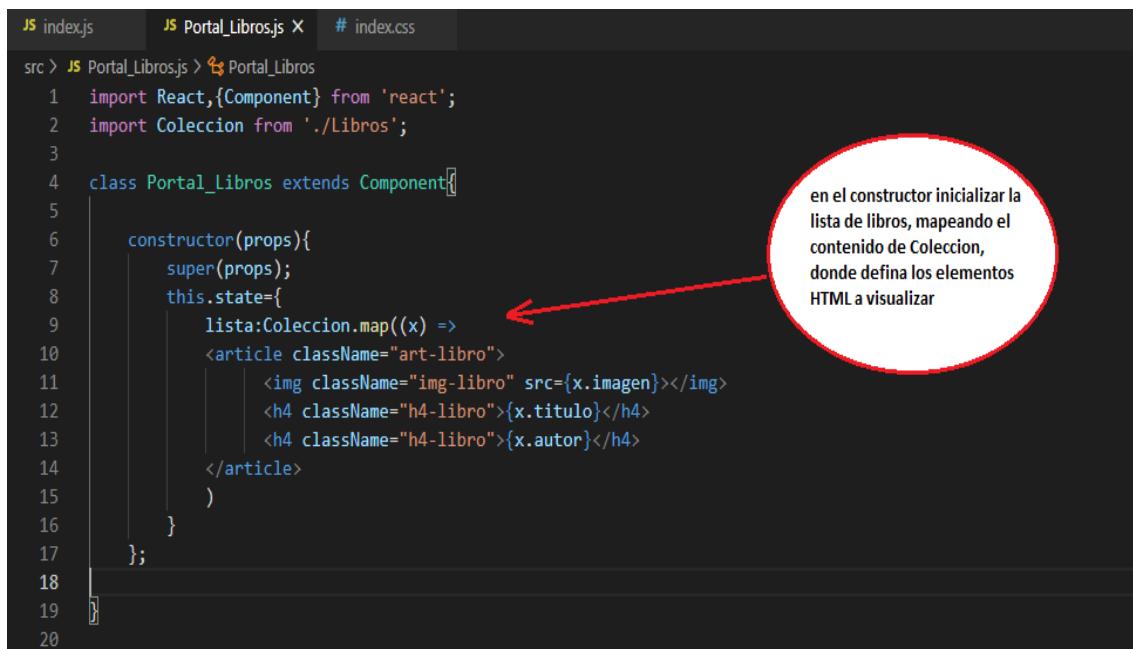
JS index.js JS Portal_Libros.js X # index.css
src > JS Portal_Libros.js > ...
1 import React,{Component} from 'react';
2 import Coleccion from './Libros';
3
4
5 class Portal_Libros extends Component{
6
7 }
8
9 export default Portal_Libros;
10
11

```

Figura 285: Portal\_Libros.js

Fuente.- Elaboración Propia

Para iniciar, definimos el constructor de la clase, el cual inicializa el contenido del estado lista, mapeando el contenido del Array Colección, generando estructura HTML.



```

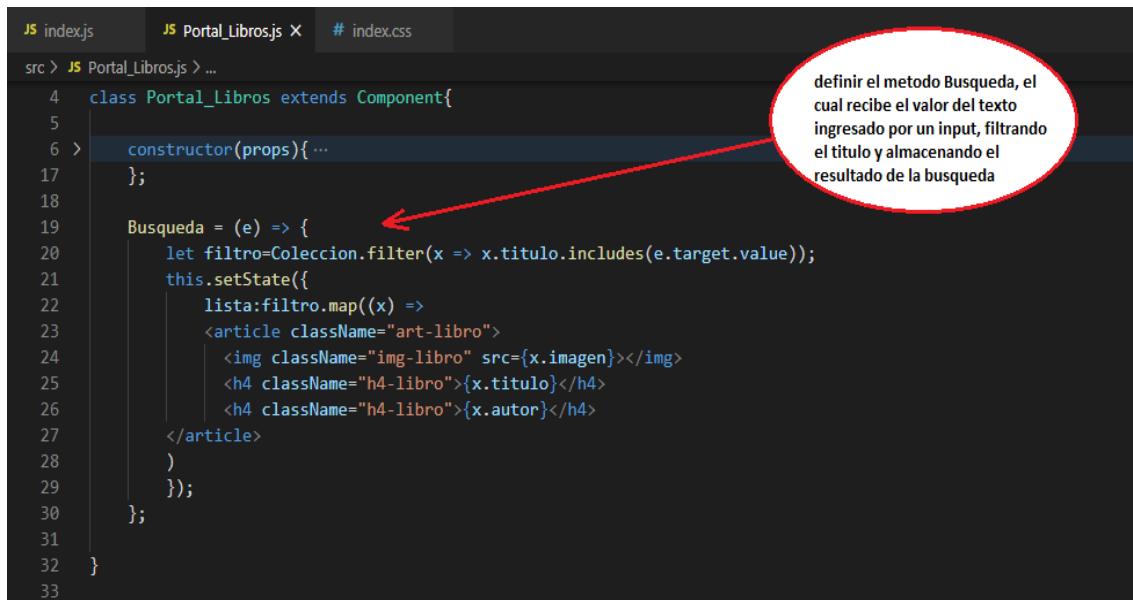
JS index.js      JS Portal_Libros.js X  # index.css
src > JS Portal_Libros.js > Portal_Libros
1 import React,{Component} from 'react';
2 import Coleccion from './Libros';
3
4 class Portal_Libros extends Component{
5
6     constructor(props){
7         super(props);
8         this.state={
9             lista:Coleccion.map((x) =>
10                 <article className="art-libro">
11                     <img className="img-libro" src={x.imagen}></img>
12                     <h4 className="h4-libro">{x.titulo}</h4>
13                     <h4 className="h4-libro">{x.autor}</h4>
14                 </article>
15             )
16         }
17     }
18 }
19 ]
20

```

en el constructor inicializar la lista de libros, mapeando el contenido de Coleccion, donde defina los elementos HTML a visualizar

Figura 286: Portal\_Libros.js  
Fuente.- Elaboración Propia

Defina el método Búsqueda, el cual filtra los elementos almacenados por Colección y el resultado se almacena en filtro.



```

JS index.js      JS Portal_Libros.js X  # index.css
src > JS Portal_Libros.js > ...
4 class Portal_Libros extends Component{
5
6     constructor(props){ ...
7     };
8
9     Busqueda = (e) => {
10         let filtro=Coleccion.filter(x => x.titulo.includes(e.target.value));
11         this.setState({
12             lista:filtro.map((x) =>
13                 <article className="art-libro">
14                     <img className="img-libro" src={x.imagen}></img>
15                     <h4 className="h4-libro">{x.titulo}</h4>
16                     <h4 className="h4-libro">{x.autor}</h4>
17                 </article>
18             )
19         });
20     };
21
22 }
23 ]
24

```

definir el metodo Busqueda, el cual recibe el valor del texto ingresado por un input, filtrando el titulo y almacenando el resultado de la busqueda

Figura 287: Portal\_Libros.js  
Fuente.- Elaboración Propia

En el render, el bloque para colocar el input donde se ingresa las iniciales del título y enlazando el evento OnChange con el método Búsqueda, defina el Array de lista, el cual visualiza los libros

```

JS index.js    JS Portal_Libros.js # index.css
src > JS Portal_Libros.js ...
4   class Portal_Libros extends Component{
5
6 >   constructor(props){ ... }
7
8
9 >   Busqueda = (e) => { ... }
10
11
12   render(){
13     return(
14       <div>
15         <h1 className="h1-libro">Literatura Peruana</h1>
16         <div className="div-buscar">
17           <span></span>
18           <input name="buscar" placeholder="Buscar por Titulo" onChange={this.Busqueda} />
19           
20         </div>
21
22         <div>
23           {this.state.lista}
24         </div>
25       </div>
26     )
27   }
28
29   export default Portal_Libros;

```

Figura 288: Portal\_Libros.js

Fuente.- Elaboración Propia

### Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar el archivo Portal\_Libros.js y lo llamaremos Portal
- Al renderizar la página importamos el contenido de Portal

```

JS index.js X
src > JS index.js
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import Social from './Header_social';
5   import Header from './Header';
6   import Portal from './Portal.Libros';
7
8   ReactDOM.render([
9     <div>
10       <Social />
11       <Header />
12       <Portal />
13     </div>,
14     document.getElementById('root')
15   ]);
16

```

Figura 289: Index.js

Fuente.- Elaboración Propia



Figura 290: Index.html  
Fuente.- Elaboración Propia

## LABORATORIO 4

### Implementando la paginación de los libros en una aplicación React JS

En este laboratorio vamos a implementar una paginación de libros, utilizando numeración definidas por etiquetas `<a>`

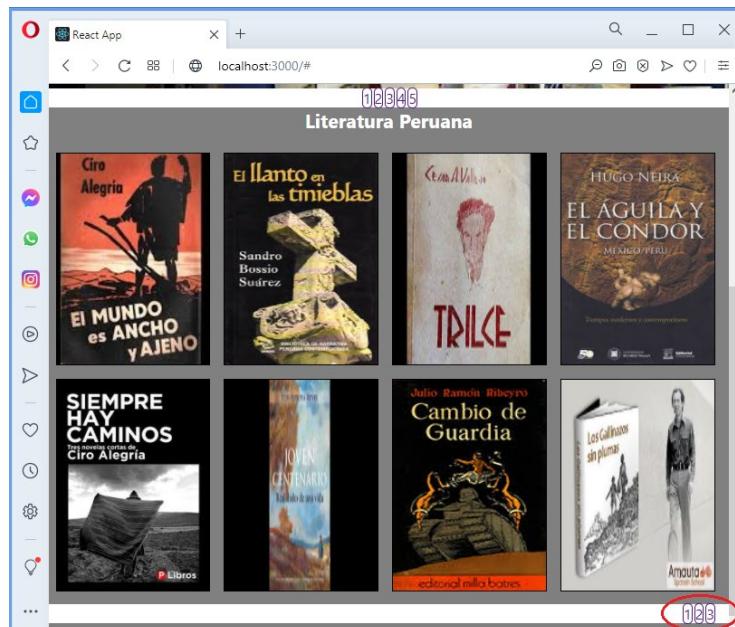


Figura 291: Diseño de paginación de libros

Fuente.- Elaboración Propia

### Agregando selectores a index.css

Primero agregamos en el archivo `index.css`, los selectores de class y definimos sus propiedades.

```

JS index.js      JS Paginar_Libros.js    # index.css X
src > # index.css > .item:hover
115
116 .div-paginar{
117   width: 98%;
118   height: auto;
119   float: left;
120   padding-right: 2%;
121   margin-bottom: 10px;
122   margin-top:10px;
123   text-align: right;
124   background-color: white;
125 }
126
127 .item{
128   font-size: 1.5em;
129   margin-left: 3px;
130   border: 1px solid;
131   text-decoration: none;
132   border-radius: 10px;
133 }
134
135 .item:hover{
136   background-color: gray;
137   color:white;
138 }
```

definir el selector de class div-paginar, el cual agrupará la numeración de la paginación

definir el selector de class item, para los numeros de la paginacion

Figura 292: Index.css

Fuente.- Elaboración Propia

### Trabajando con Paginar\_Libros.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Paginar\_Libros.js.

- En el archivo importar React y {Component}
- Importamos la colección de libros el cual llamaremos colección
- A continuación, defina la clase Paginar\_Libros de tipo Component

```

JS index.js      JS Paginar_Libros.js X
src > JS Paginar_Libros.js > ...
1 import React,{Component} from 'react';
2 import Coleccion from './Libros';
3
4 class Paginar_Libros extends Component{
5
6 }
7
8 export default Paginar_Libros;

```

Figura 293: Paginar\_Libros.js

Fuente.- Elaboración Propia

Para iniciar, definimos un Array de números llamado np, el cual estará conformado por el conjunto de números de la paginación (8 libros por página). Utilizamos Colección.length para dicho proceso

```

JS index.js      JS Paginar_Libros.js X
src > JS Paginar_Libros.js > ...
1 import React,{Component} from 'react';
2 import Coleccion from './Libros';
3
4 let np=new Array();
5 for( let i=1; i<= (Coleccion.length%8==0 ? Coleccion.length/8 : Coleccion.length/8 +1 );i++)
6 {
7     np.push(i);
8 }
9
10 > class Paginar_Libros extends Component{ ...
55 }
56

```

Figura 294: Paginar\_Libros.js

Fuente.- Elaboración Propia

Para iniciar, definimos en el constructor de la clase, dos estados: lista, la cual almacena los 8 primeros elementos de Colección, y el estado numeral, el cual almacena la colección de hipervínculos para la numeración de la paginación y asociamos el evento Click con el método Paginar.

```

JS index.js      JS Paginar_Libros.js X
src > JS Paginar_Libros.js > ...
10  class Paginar_Libros extends Component{
11
12    constructor(props){
13      super(props);
14      this.state={
15        lista:Coleccion.slice(0,8).map((x) =>
16          <article className="art-libro">
17            <img className="img-libro" src={x.imagen}></img>
18            <h4 className="h4-libro">{x.titulo}</h4>
19            <h4 className="h4-libro">{x.autor}</h4>
20          </article>
21        ),
22
23        numeral:np.map((x) =>
24          <a href="#" className="item" data-op={x-1} onClick={this.Paginar}>{x}</a>
25        ),
26      }
27    };
28  }
29 }

```

Figura 295: Paginar\_Libros.js

Fuente.- Elaboración Propia

Defina el método Paginar, el cual muestra los elementos almacenados por Colección desde un rango: múltiplo del número de página por 8 y el fin la suma de 8: slice(n,m)

```

JS index.js      JS Portal_Libros.js X
src > JS Portal_Libros.js > ...
10  class Paginar_Libros extends Component{
11
12  >  constructor(props){ ...
13  };
14
15  Paginar= (e) => {
16    let n=parseInt(e.target.dataset.op)*8;
17    let m=n+8;
18    this.setState({
19      lista:Coleccion.slice(n,m).map((x) =>
20        <article className="art-libro">
21          <img className="img-libro" src={x.imagen}></img>
22          <h4 className="h4-libro">{x.titulo}</h4>
23          <h4 className="h4-libro">{x.autor}</h4>
24        </article>
25      )
26    });
27  }
28
29 }

```

Figura 296: Portal\_Libros.js

Fuente.- Elaboración Propia

En el render, agregamos los estados lista (Array de Colección) y el estado numeral (Array de hipervínculos)

```

JS index.js    JS Paginar_Libros.js X
src > JS Paginar_Libros.js > ...
10  class Paginar_Libros extends Component{
11
12 >   constructor(props){ ... }
13
14
15
16
17
18
19 >   Paginar= (e) => { ... }
20
21
22
23
24
25
26
27
28
29 >   render(){
30     return(
31       <div>
32         <h1 className="h1-libro">Literatura Peruana</h1>
33         <div>{this.state.lista}</div>
34         <div className="div-paginar">{this.state.numeral}</div>
35       </div>
36     )
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54 }

```

Figura 297: Portal\_Libros.js

Fuente.- Elaboración Propia

### Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar el archivo Paginar\_Libros.js y lo llamaremos Portal
- Al renderizar la página importamos el contenido de Portal

```

JS index.js X  JS Paginar_Libros.js
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import Social from './Header_social';
5  import Header from './Header';
6  import Portal from './Paginar_Libros';
7
8  ReactDOM.render(
9    <div>
10      <Social />
11      <Header />
12      <Portal />
13    </div>,
14    document.getElementById('root')
15  );
16

```

Figura 298: Index.js

Fuente.- Elaboración Propia

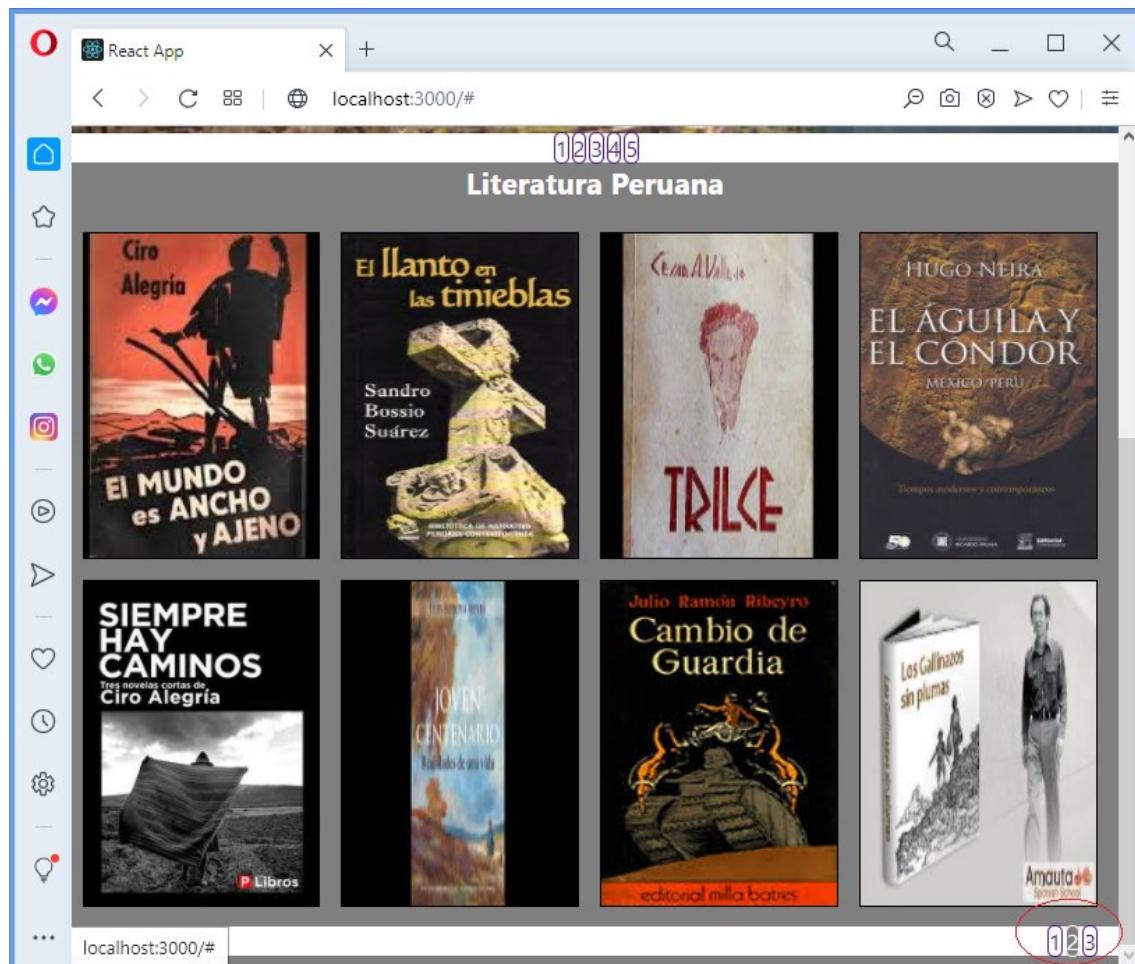


Figura 299: Index.html  
Fuente.- Elaboración Propia

## LABORATORIO 5

### Implementando el ordenamiento de los libros en una aplicación React JS

En este laboratorio vamos a implementar el ordenamiento de los libros al hacer click a una de las columnas: id, título o autor.

	Ordenar	Ordenar
ID	Título	Autor
15	Cambio de guardia	Ramon Ribeyro
7	Cuentos Andinos	Lopez Albujar
17	Cuentos y ensayos	Ramon Ribeyro
12	El aguila y el condor	Hugo Neira
2	El caballero carmelo	Valderomar
9	El mundo es ancho y ajeno	Ciro Alegria
16	Gallinazos sin Plumas	Ramon Ribeyro
14	Joven Centenario	Bedoya Reyes
6	La chunga	Vargas Llosa
20	La ciudad y los perros	Vargas Llosa
4	La decima en el Peru	Nicomedes Santa Cruz
5	La llamada de la Tribu	Vargas Llosa
18	La palabra del mudo	Ramon Ribeyro

Figura 300: Diseño de ordenamiento de libros

Fuente.- Elaboración Propia

### Agregando selectores a index.css

Primero agregamos en el archivo index.css, los selectores de class y definimos sus propiedades

```
JS Listar_Libros.js  JS Paginar_Libros.js  JS Libros_Portal.js  # index.css  JS index.js
src > # index.css > ...
146 .portal{
147   width: 100%;
148   float: left;
149   background-color: white;
150 }
151 .fila{
152   width: 100%;
153   margin-bottom: 2px;
154   float: left;
155 }
156 .celda1{
157   width: 5%;
158   float: left;
159   text-align: center;
160 }
161 .celda2{
162   width: 40%;
163   float: left;
164 }
165 .celda4{
166   width: 15%;
167   float: left;
168 }
169 .img-book{
170   width: 50px;
171   height: 50px;
172 }
```

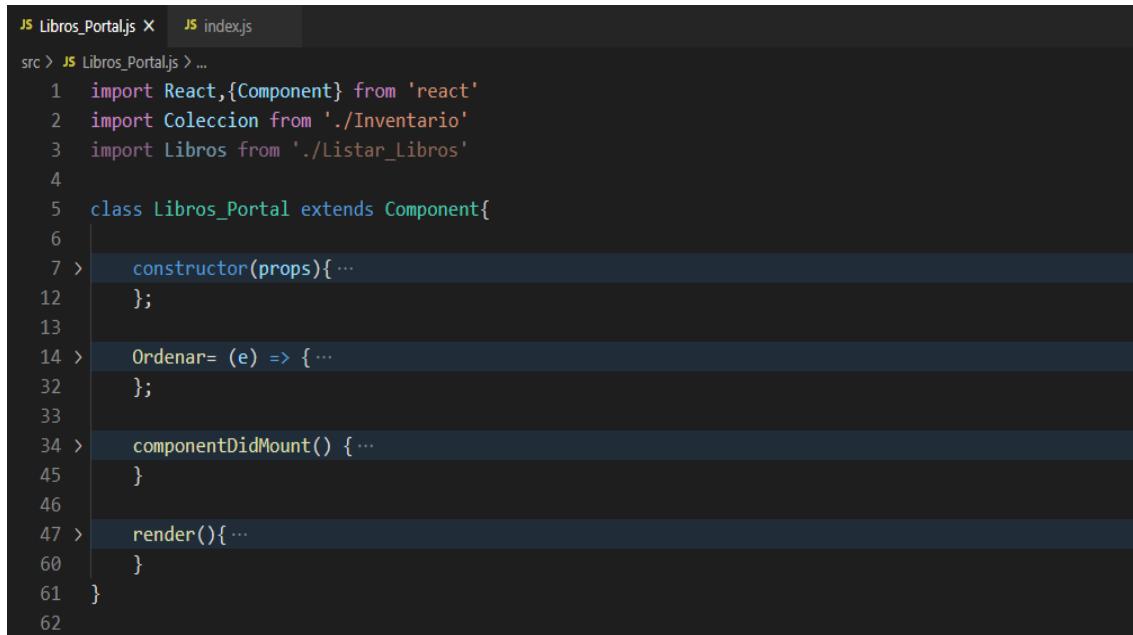
Figura 301: Index.css

Fuente.- Elaboración Propia

### Trabajando con Libros\_Portal.js

En la carpeta src, agregamos un nuevo Archivo (new File) y lo llamaremos Libros\_Portal.js.

- En el archivo importar React, Colección de Inventario y Libros de Listar\_Libros
- Defina una clase de tipo Component



```

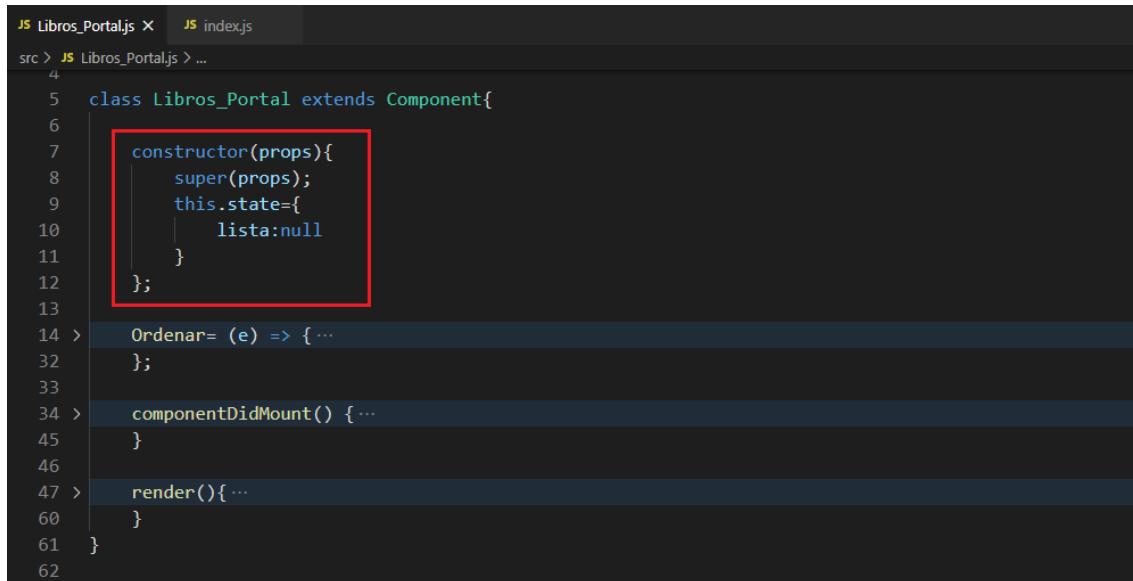
JS Libros_Portal.js X JS index.js
src > JS Libros_Portal.js > ...
1  import React,{Component} from 'react'
2  import Coleccion from './Inventario'
3  import Libros from './Listar.Libros'
4
5  class Libros_Portal extends Component{
6
7    >   constructor(props){ ...
8      ...
9
10   >  Ordenar= (e) => { ...
11     ...
12
13   >  componentDidMount() { ...
14     ...
15
16   >  render(){...
17     ...
18
19   }
20
21 }
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62

```

Figura 302: Libros\_Portal.js

Fuente.- Elaboración Propia

Para iniciar, definimos en el constructor de la clase, el estado: lista donde le asignamos el valor de null



```

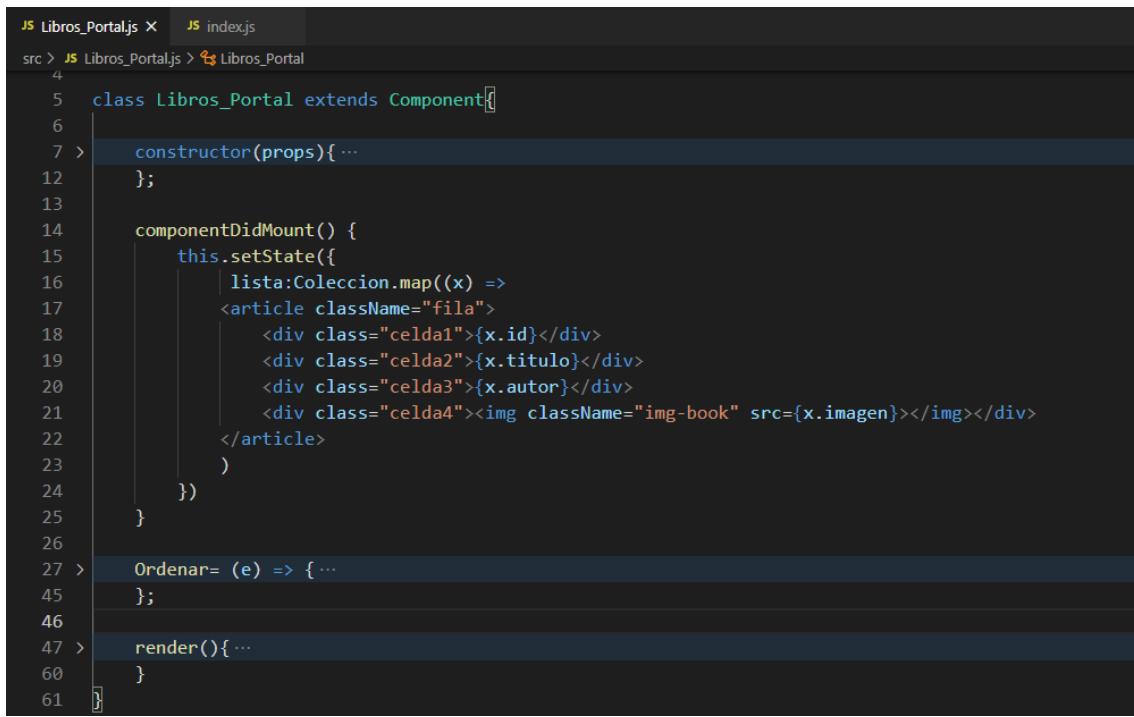
JS Libros_Portal.js X JS index.js
src > JS Libros_Portal.js > ...
4
5  class Libros_Portal extends Component{
6
7    > constructor(props){
8      super(props);
9      this.state={
10        ...
11      }
12    };
13
14   > Ordenar= (e) => { ...
15     ...
16
17   > componentDidMount() { ...
18     ...
19
20   > render(){...
21     ...
22
23   }
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62

```

Figura 303: Libros\_Portal.js

Fuente.- Elaboración Propia

Defina el método componentDidMount, para asignar al estado lista, el contenido de Colección



```

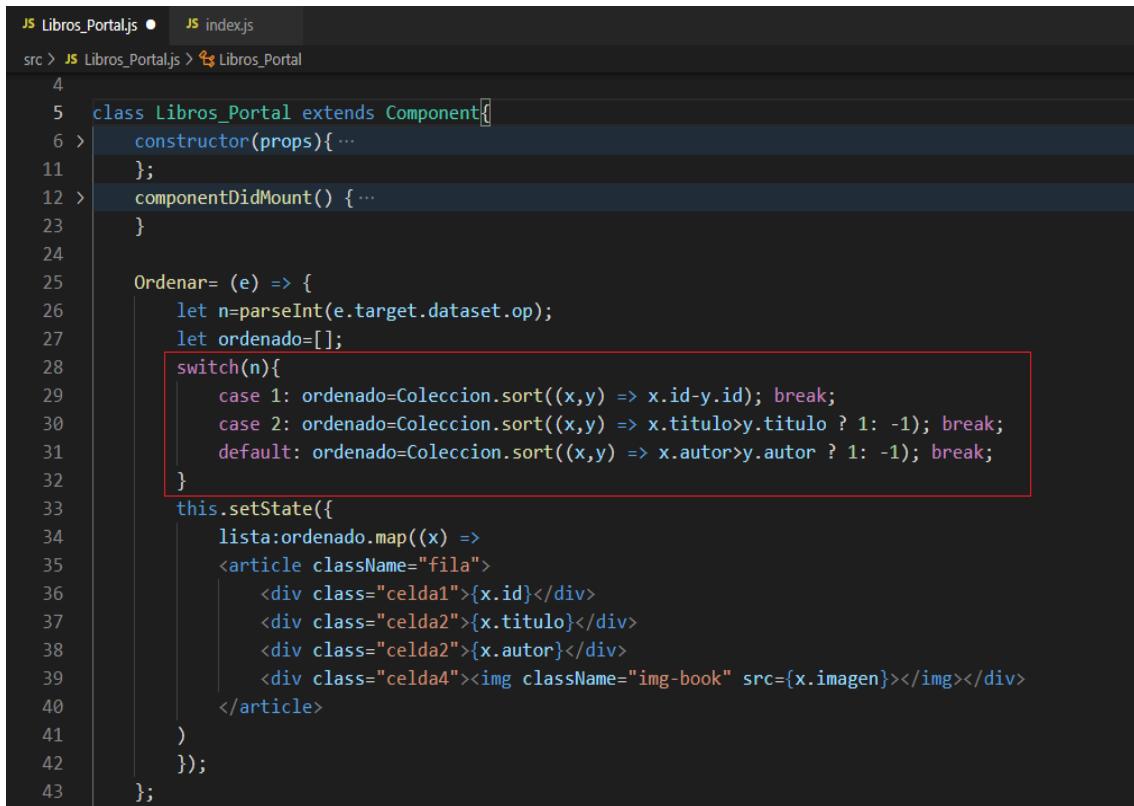
JS Libros_Portal.js ✘ JS index.js
src > JS Libros_Portal.js > ↗ Libros_Portal
  4
  5   class Libros_Portal extends Component[{
  6     constructor(props){ ...
  12   };
  13
  14   componentDidMount() {
  15     this.setState({
  16       lista:Coleccion.map((x) =>
  17         <article className="fila">
  18           <div className="celda1">{x.id}</div>
  19           <div className="celda2">{x.titulo}</div>
  20           <div className="celda3">{x.autor}</div>
  21           <div className="celda4"><img className="img-book" src={x.imagen}></img></div>
  22         </article>
  23       )
  24     })
  25   }
  26
  27   Ordenar= (e) => { ...
  45   };
  46
  47   render(){ ...
  60   }
  61 }

```

Figura 304: Libros\_Portal.js

Fuente.- Elaboración Propia

Defina el método Ordenar, el cual ordena según la opción seleccionada de <a>, retornando la lista ordenada según el campo.



```

JS Libros_Portal.js ● JS index.js
src > JS Libros_Portal.js > ↗ Libros_Portal
  4
  5   class Libros_Portal extends Component[{
  6     constructor(props){ ...
  11   };
  12   componentDidMount() { ...
  23   }

  25   Ordenar= (e) => {
  26     let n=parseInt(e.target.dataset.op);
  27     let ordenado=[];
  28     switch(n){
  29       case 1: ordenado=Coleccion.sort((x,y) => x.id-y.id); break;
  30       case 2: ordenado=Coleccion.sort((x,y) => x.titulo>y.titulo ? 1: -1); break;
  31       default: ordenado=Coleccion.sort((x,y) => x.autor>y.autor ? 1: -1); break;
  32     }
  33     this.setState({
  34       lista:ordenado.map((x) =>
  35         <article className="fila">
  36           <div className="celda1">{x.id}</div>
  37           <div className="celda2">{x.titulo}</div>
  38           <div className="celda2">{x.autor}</div>
  39           <div className="celda4"><img className="img-book" src={x.imagen}></img></div>
  40         </article>
  41       );
  42     });
  43   }

```

Figura 305: Libros\_Portal.js

Fuente.- Elaboración Propia

Al renderizar la clase, definimos los hipervínculos para definir el ordenamiento y lo enlazamos al evento Click=Ordenar.

```

JS Libros_Portal.js × JS index.js
src > JS Libros_Portal.js > ...
4
5   class Libros_Portal extends Component{
6     constructor(props){ ... }
7     ;
8     componentDidMount() { ... }
9     }
10    Ordenar= (e) => { ... }
11    ;
12    }
13    }
14    render(){
15      return(
16        <div className="portal">
17          <h1 className="h1-libro">Literatura Peruana</h1>
18          <div className="fila">
19            <a href="#" data-op="1" onClick={this.Ordenar} className="celda1">Ordenar</a>
20            <a href="#" data-op="2" onClick={this.Ordenar} className="celda2">Ordenar</a>
21            <a href="#" data-op="3" onClick={this.Ordenar} className="celda3">Ordenar</a>
22          </div>
23          <div>{this.state.lista}</div>
24        </div>
25      )
26    }
27  }
28  }
29  }
30  }
31  }
32  }
33  }
34  }
35  }
36  }
37  }
38  }
39  }
40  }
41  }
42  }
43  }
44  }
45  }
46  }
47  }
48  }
49  }
50  }
51  }
52  }
53  }
54  }
55  }
56  }
57  }

```

Figura 306: Libros\_Portal.js

Fuente.- Elaboración Propia

## Renderizar index.js

A continuación, abrir el archivo index.js y definir el siguiente código:

- Importar el archivo Paginar\_Libros.js y lo llamaremos Portal
- Al renderizar la página importamos el contenido de Portal

```

JS index.js × JS Paginar.Libros.js
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import Social from './Header_social';
5  import Header from './Header';
6  import Portal from './Paginar.Libros';
7
8  ReactDOM.render(
9    <div>
10      <Social />
11      <Header />
12      <Portal />
13    </div>,
14    document.getElementById('root')
15  );
16

```

Figura 307: Index.js

Fuente.- Elaboración Propia

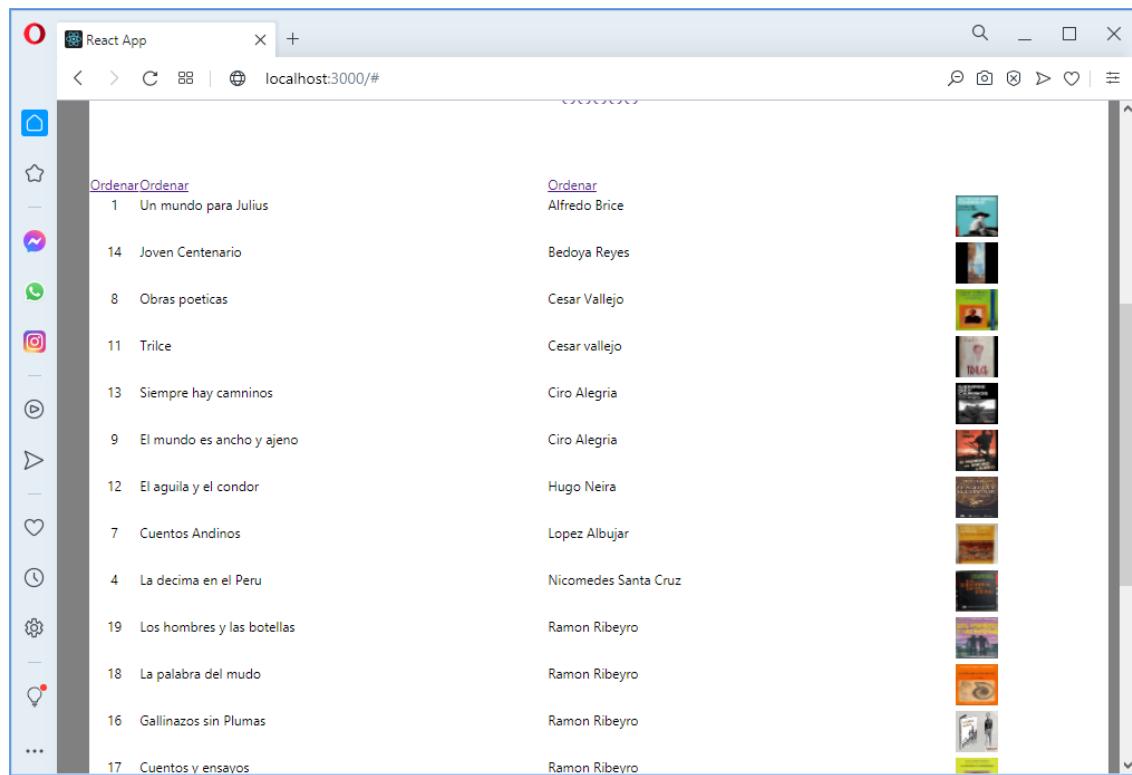


Figura 308: Index.html  
Fuente.- Elaboración Propia

# Resumen

1. El ciclo de vida no es más que una serie de estados por los cuales pasa todo componente a lo largo de su existencia. Esos estados tienen correspondencia en diversos métodos, que podemos implementar para realizar acciones cuando se van produciendo.
2. Un ciclo de vida de un componente está clasificado en tres estados: montaje, actualización y desmontaje.
3. React tiene un sistema de eventos sintéticos que ejecutan una acción cuando ocurre un acontecimiento. Las declaraciones de eventos en React tienen siempre esa forma: el prefijo "on", seguido del tipo de evento que queremos capturar (onClick, onInput...).
4. El renderizado condicional en React funciona de la misma forma que lo hacen las condiciones en JavaScript. Usa operadores de JavaScript como if o el operador condicional para crear elementos representando el estado actual, y deja que React actualice la interfaz de usuario para emparejarlos.
5. Para la creación de elementos HTML se realiza mayormente usando una herramienta del propio lenguaje JavaScript, el método **map()** disponible en los Arrays.
6. La idea bajo React es crear componentes reusables. Si varios de nuestros componentes poseen la misma funcionalidad, podemos crear un módulo y reusarlo en todos ellos. Los **mixins** nos permiten hacer eso.

# Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://desarrolloweb.com/articulos/estado-componentes-react.html>
- <https://es.reactjs.org/docs/state-and-lifecycle.html>
- <https://mauriciogc.medium.com/3-react-ciclo-de-vida-de-un-componente-propiedades-y-estado-a1b1d9965af1>
- [https://books.adalab.es/materiales-front-end-d/sprint-3.-es6-y-react/3\\_7\\_eventos\\_react](https://books.adalab.es/materiales-front-end-d/sprint-3.-es6-y-react/3_7_eventos_react)
- [https://books.adalab.es/materiales-front-end-d/sprint-3.-es6-y-react/3\\_7\\_eventos\\_react#uso-de-metodos-handleevent-para-manejar-eventos](https://books.adalab.es/materiales-front-end-d/sprint-3.-es6-y-react/3_7_eventos_react#uso-de-metodos-handleevent-para-manejar-eventos)
- <https://es.reactjs.org/docs/faq-functions.html>
- <https://desarrolloweb.com/articulos/condicionales-templates-javascript-react.html>
- <https://es.reactjs.org/docs/conditional-rendering.html>
- <https://desarrolloweb.com/articulos/condicionales-templates-javascript-react.html>
- <https://desarrolloweb.com/articulos/repeticiones-templates-javascript-react.html>
- <https://abalozz.es/reutiliza-tu-codigo-en-react-usando-mixins/>

# Bibliografía

- Álvarez, Alonso (2012) *Manual imprescindible de HTML 5*. Madrid: Anaya Multimedia.  
Centro de Información: Código 006.74HTML ALVA/M
- Álvarez García, Alonso (2019) *Manual Imprescindible: HTML 5.2*. Madrid: Anaya Multimedia.  
Centro de Información: Código 006.74HTML ALVA/M
- Cunlimón (2020) *El gran libro de HTML5, CSS3 y JavaScript*. Recuperado de [http://www.cunlimon.ac.cr/Uploads/InfoPublica/HTML5\\_CSS3\\_JavaScript.pdf](http://www.cunlimon.ac.cr/Uploads/InfoPublica/HTML5_CSS3_JavaScript.pdf)
- Goldstein, Alexis (2011) *Manual imprescindible de HTML 5 y CSS3*. Madrid: Anaya Multimedia.  
Centro de Información: Código 006.74 HTML GOLD
- Gauchar, J. D. (2019) *El gran libro de HTML5, CSS3 y JavaScript*. 3a ed. Marcombo.  
Recuperado de <https://elevaciondigital.pe/wp-content/uploads/2019/06/El-gran-libro-de-HTML5-CSS3-y-JavaScript.pdf>
- Lancker, Luc Van (2014) *jQuery. El Framework de JavaScript de la Web 2.0*. 2a ed. Barcelona: ENI.  
Centro de Información: Código 005.133JS LANC 2014
- Luján Castillo, José Dimas (2016) *HTML5, CSS y JAVASCRIPT: crea tu web y apps con el estándar de desarrollo*. Bogotá: Alfaomega.  
Centro de información: 006.74HTML LUJA
- MacDonald, Matthew (2016) *Creación y diseño web*. 4a ed. Madrid: Anaya Multimedia.  
Centro de Información: Código 006.7 MACD 2016
- McFarland, David (2015) *JavaScript y jQuery*. 3a ed. Madrid: Anaya Multimedia.  
Centro de Información: Código 005.133JS MCFA/J 2016
- Meloni, Julie C. (2012) *HTML5, CSS3 y JavaScript*. Madrid: Anaya Multimedia.  
Centro de Información: Código 006.74HTML MELO
- Meloni, Julie C. (2015) *HTML5, CSS3 y JavaScript*. Madrid: Anaya Multimedia.  
Centro de Información: Código 006.74HTML MELO 2015
- Rubiales Gómez, Mario (2018) *Curso de desarrollo web: HTML, CSS y JavaScript*. Madrid: Anaya Multimedia  
Centro de Información: Código 006.74HTML RUBI/C
- Desarrollo Web (s.f.) *Manual de React JS*. Recuperado de <https://desarrolloweb.com/manuales/manual-de-react.html>
- Uniwebsidad (2021) *Libros Web*. Recuperado de <http://librosweb.es/libros/>