Carrito de la compra

Desarrollo de Interfaces

Israel Serrano

Pablo Baldazo

Jorge Buendía

Desarrollo de Interfaces

2ºDAM

Información del documento

Nombre del documento	Carrito de la compra
Autor del documento	Israel Serrano, Pablo Baldazo, Jorge Buendía
Versión del proyecto	v1.0
Fecha de entrega	20.10.2022
Repositorio GitHub	https://github.com/PIJ-Group/ShopingCart.git

Distribución y autores del documento final

Nombre	Organización/Puesto
Israel Serrano	Cada uno de los integrantes ha realizado el HTML y el CSS de manera individual, posteriormente hemos puesto en común analizando y trabajando en la solución final para esas dos partes. Para la parte de JavaScript nos repartimos las funcionalidades a realizar, comentando cada paso con el resto de los compañeros para solucionar los problemas que nos íbamos encontrando y tomar decisiones en equipo.
Pablo Baldazo	
Jorge Buendía	

Contenido

1 Introducción

1.1 Objetivo

El objetivo de este proyecto es el de crear una página web dinámica que simule un carrito de la compra utilizando lo aprendido el curso pasado de HTML5 y CSS3, junto con los nuevos conocimientos de JavaScript aprendidos en el curso actual.

2 Requerimientos

Requerimiento 1

Crea una página web dinámica usando JavaScript que implemente las siguientes especificaciones:

- Constará de un formulario para recoger los artículos de un carrito de compra.
- El botón “Añadir al carrito” debe tener la funcionalidad siguiente: cuando el usuario rellene las tres primeras cajas de texto (artículo, precio y unidades) y haga clic sobre dicho botón, deberá sumar el precio del artículo por el número de unidades al “Precio total carrito” de los artículos anteriormente añadidos y el nombre del artículo deberá añadirse a “Artículos en el carrito”, de modo que finalmente se puede ver la lista total de la compra con todos los productos y el precio total de dicha compra o carrito.

  Cada vez que se pulsa, además, se vuelven a resetear las tres primeras cajas para poder introducir un nuevo producto, el número de unidades será de 1 por defecto. La caja para el nombre del artículo recibe el foco para facilitar la entrada de datos. Solo se podrán añadir ítems al carrito si se completan tanto el artículo como su precio. Si alguno de los dos faltase al hacer clic sobre el botón “Añadir al carrito”, se mostraría un texto al lado de la caja correspondiente advirtiéndolo, además se haría el foco de nuevo en la caja.

  También es necesario validar el precio del artículo para que solo acepte datos numéricos, sino se advertirá de la misma forma que la anterior al hacer clic en el botón “Añadir al carrito”.

- Habrá dos formas de pagar:
    - Si se selecciona “Tarjeta” aparecerán tres nuevas cajas de texto para introducir los datos de la tarjeta bancaria.
    - Si se selecciona “Efectivo” aparecerá una nueva caja de texto con el importe total del carrito.
- El botón “Imprimir” se habilitará cuando se acepten las condiciones de compra.
- Cuando se pulse el botón “Imprimir” debe mostrar a través de una ventana, tanto la lista de la compra final como el precio final del carrito.
- Si no se ha seleccionado una forma de pago deberá aparecer el mensaje “Seleccione una forma de pago”.
- Con el botón “Restablecer” se resetean todas las cajas de texto para poder introducir un nuevo producto. La caja para el nombre del artículo recibe el foco para facilitar la entrada de datos. Inicializa la caja “Precio total del carrito” a 0 para facilitar las operaciones aritméticas.

Requerimiento 2

Se pide:

1-Escribe el contenido HTML de la página web.

2- Realiza toda la lógica del programa en un archivo JavaScript externo.

3- Aplica estilos mediante una hoja CSS externa.

Se valorará una buena presentación de la página web, pero lo importante es la parte funcional de JavaScript. Es fundamental que durante la ejecución de la aplicación web no se produzcan excepciones ni ningún tipo de error. Habrá que gestionarlos y tenerlos en cuenta desde el código.

En todo momento habrá que informar al usuario de las necesidades del programa.

3 Explicación de la actividad

Para la realización de este proyecto, hemos modularizado el trabajo a partir de tres grandes archivos:

3.1 HTML

El primero de ellos contiene el html de dicho proyecto, este archivo es el “esqueleto” a partir del cual empezamos a trabajar y en el que definimos todos aquellos elementos que van a ser visibles a ojos de los usuarios cuando entren al sitio web y con el que podrán interactuar una vez le otorguemos la funcionalidad deseada.

En los siguientes fragmentos de código explicaremos los elementos y puntos más importantes de nuestro archivo “**index.html**”:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tarea AE-1</title>
    <link rel="icon" href="img/cesta_pestaña.png">
    <link href="css/estilos.css" rel="stylesheet">
    <script src="js/cart.js"></script>
</head>
```

En la primera sección de nuestro archivo html, enlazamos a través de las etiquetas <link> y <script> una hoja de estilos CSS y un archivo JavaScript respectivamente, los cuales detallaremos más adelante en sus correspondientes secciones.

```
<body>
<header>
    <img src="img/logo.PNG" width="400" alt="Icono supermercado" id="logo">
</header>
<main>
  <section>
    <header>
                <h2>TU CESTA</h2>
    </header>
      <article id="form">
       <form action="procesarPeticion.jsp" method="get">
         <label for="articleName">Nombre artículo: </label>
          <input id="articleName" type="text" name="articleName" autofocus>
            <span id="spanName"></span>

         <label for="articlePrice">Precio artículo: </label>
          <input id="articlePrice" type="text" name="articlePrice">
            <span id="spanArticlePrice"></span>

          <label for="units">Unidades: </label>
           <input id="units" type="number" min="1" value="1" name="units">

           <input id="add" type="button" value="Añadir al carrito">

          <label for="cartArticles">Artículos en el carrito:</label>
           <input id="cartArticles" type="text" name="cartArticles"
            size="80"readonly>

          <label for="totalPrice">Precio total del carrito: </label>
           <input id="totalPrice" type="text" name="totalPrice" readonly>
```

A partir de este punto ya nos centramos en el cuerpo del html.

Iniciamos el formulario de la página a partir de la etiqueta <form>, donde vamos anidando dentro del mismo, varias etiquetas <label> y sus correspondientes <input> que hacen referencia a campos claves del formulario, que son designados exclusivamente a través de atributos "**id**" para poder tener control sobre ellos más adelante en nuestro archivo JavaScript.

Observamos que hay una etiqueta <input> de tipo “**button**”, que de momento tiene una funcionalidad nula, pero que posteriormente a través de un evento en el archivo JavaScript, ese botón será capaz de añadir artículos a nuestro carrito de la compra, por tanto, le hemos asignado también un “**id**”.

Cabe destacar, la utilización de etiquetas , que posteriormente utilizaremos para mostrar mensajes de error, o mandar advertencias al usuario si este no ingresa ningún texto en los campos del formulario, o si lo hace de manera errónea.

También se han utilizado algunos atributos interesantes, como “autofocus” para que el puntero se encuentre automáticamente en el primer campo del formulario y el atributo “readonly” para que los campos relacionados con el precio total de la suma de productos no puedan ser modificados por el usuario durante la interacción con el formulario.

```
<label id="wayToPay" class="wayToPay" for="wayToPay">Forma de pago:
</label>
<select id="selectWayToPay" class="selectWayToPay" name="wayToPay">
      <option value="S">--Seleccione--</option>
      <option value="T">Tarjeta</option>
      <option value="E">Efectivo</option>
</select>

<div id="cardPay" class="cardPay">
      <label for="cardHolder">Titular tarjeta: </label>
       <input id="cardHolder" type="text" name="cardHolder">

      <label for="cardNumber">Número tarjeta: </label>
       <input id="cardNumber" type="text" name="cardNumber" maxlength="16">

      <label for="cvv">CVV: </label>
       <input id="cvv" type="password" name="cvv" maxlength="3" size="3">
</div>

<div id="cashPay" class="cashPay">
      <label for="cashAmount">Importe efectivo: </label>
       <input id="cashAmount" type="text" name="cashAmount" readonly>
      <label for="cashDelivered">Efectivo entregado: </label>
       <input id="cashDelivered" type="text" name="cashDelivered">
</div>
```

```
<input class="buttonTerms" id="terms" type="checkbox" name="terms"
 value="Y">
<label class="terms" for="terms">Acepto las condiciones de la
 compra</label>

 <a id="tc"ref="docs/Terminos_de_uso_y_condiciones_generales_de_compra.pdf"
 target="_blank">Condiciones de compra</a>

 <input class="print" id="print" type="button" value="Imprimir">

 <input class="reset" id="reset" type="button" value="Restablecer">
</form>
```

En la siguiente sección del formulario, podemos observar una etiqueta <select>, la cual representa un control que permite elegir al usuario entre las diferentes opciones de pago. Todas ellas tienen asignado un valor, que más adelante utilizaremos para modificar el comportamiento del formulario en función de la opción que elija el usuario.

Otro elemento a destacar, son las etiquetas que engloban los métodos de pago (tarjeta o efectivo), están designados unívocamente con un "**id**" para obtener el control total de estos campos y poder desplegarlos más adelante cuando se escriba el código correspondiente en el archivo JavaScript.

Antes del cierre del formulario, tenemos un "**checkbox**" que se traduce en unos términos y condiciones que el usuario debe aceptar antes de imprimir la compra. Si no se aceptan los términos, el botón de impresión estará permanentemente bloqueado. Esta acción también podrá ser implementada de varias formas, por tanto, asignamos tanto al checkbox como al botón de imprimir nuevamente un "id".

Finalmente, el formulario termina con un último botón, que recarga la página en caso de que el usuario quiera empezar de nuevo. Dicho botón también implementará esta funcionalidad en el archivo JavaScript.

3.2 CSS

El segundo archivo sería la hoja de estilos css "**estilos.css**":

La paleta de colores escogida para la implementación en la página es la siguiente:

Palette Number 58 | Color Palettes

Utilizando los siguientes colores para cada propósito:

#FBB040: para los resaltados de la página al tener un color similar al del nombre de la tienda.

#506F86: para el fondo de la página en cuestión.

#2F3C4F: para el fondo del pie de página.

Además de estos colores como paleta principal, se usará el color "black" para el fondo de la cabecera, al tener un color similar al fondo del logotipo empresarial, `#f9f9f9b9 (gris claro)` para el fondo de los inputs, el color "gray" para las características bloqueadas y el color `#fcfcfce3 (gris claro)` como base del formulario para complementarlo con el difuminado y así obtener el resultado deseado.

A continuación, repasamos los puntos más importantes de los estilos de la página:

Con el fin de prevenir comportamientos erróneos por parámetros prefijados en los navegadores, implementamos un estilo general a la página para ajustar los bordes, márgenes y padding:

```
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}
```

El siguiente paso será la colocación y coloración de la cabecera de la página, que contendrá el logotipo empresarial, y cuyo fondo será del mismo tono que dicho logotipo para lograr homogeneidad:

```
#logo{
    display: block;
    margin: 0 auto;
    padding-bottom: 10px;
}

header {
    background-color: black;
    margin-bottom: 51px;
    padding-top: 15px;
}
```

Una vez realizado los estilos de la cabecera, procedemos a realizar unos estos muy simples para todo el cuerpo de la página, dotándole de color y asegurándonos que las proporciones llegarán al máximo de la página:

```
body {
    background-color: #506F86;
    width: 100%;
    height: 100%;
}
```

Procedemos a realizar los estilos del formulario, parte principal del proyecto, para lo cual lo separamos en dos partes, la cabecera del propio formulario y su cuerpo.

En el caso de la cabecera, además de centrar su texto, tanto aquí como en el cuerpo dotaremos al color de fondo de un degradado y una sombra para ganar en volumen y lograr una diferencia con el fondo utilizando prácticamente el mismo color:

```
section>header {
    background: linear-gradient(145deg, #506F86, #2F3C4F);
    box-shadow:  10px 10px 20px #2F3C4F;
    background-color: #fcfcfce3;
    border-top-left-radius: 10px;
    border-top-right-radius: 10px;
    border-bottom: solid 1px grey;
    width: 750px;
    margin: 0 auto;
    padding: 20px 0px 20px 0px;
}

h2 {
    text-align: center;
    font-family: system-ui, -apple-system;
    font-size: 30px;
    -webkit-text-stroke: 2px black;
    color: #FBB040;
}

form {
    background: linear-gradient(145deg, #506F86, #2F3C4F);
    box-shadow:  10px 10px 20px #2F3C4F;
    position: relative;
    background-color: #fcfcfce3;
    border-bottom-left-radius: 10px;
    border-bottom-right-radius: 10px;
    width: 750px;
    margin: 0 auto;
    margin-bottom: 74px;
    padding: 30px 450px 20px 30px;
}
```

Dentro del formulario, formatearemos todas las etiquetas y casillas para poder lograr un aspecto amplio y asegurarnos que ningún mensaje de alerta entre en conflicto con otras partes del texto, además de conseguir una mayor calidad de lectura:

- Colocación de inputs y sus etiquetas:

```
form label {
    display: block;
    margin-bottom: 5px;
    color: white;
}

form input {
    clear: right;
    margin-bottom: 10px;
    border-radius: 5px;
    border-color: #FBB040;
    background-color: #f9f9f9b9;
    padding: 1px 5px 1px 5px;

}
```

- Colocación y coloración de las cajas de advertencia:

```
#spanName{
    position: absolute;
    color: white;
    margin-left: 10px;
}

#spanArticlePrice{
    position: absolute;
    color: white;
    margin-left: 10px;
}
```

- Ejemplo de uso de hover, para cambiar el puntero cuando pasamos por casillas a las que podemos “clickar” (botón de imprimir y resetear, el check de Términos y condiciones, etc):

```
#add:hover {
    cursor: pointer
}
```

- Estilos para cambiar la decoración del enlace a las condiciones de compra, para poderle otorgar un aspecto más amigable con el escogido en el resto de la página:

```
#tc {
    display: block;
    margin-bottom: 15px;
    color: #FBB040;
    text-decoration: none;
}
```

Por último, se realizarán los estilos del pie de página, unos estilos sencillos para poder utilizar la paleta de colores escogida al completo:

```
footer {
    position: relative;
    width: 100%;
    height: 55px;
    background-color: #2F3C4F;
    text-align: right;
    padding: 15px 30px 15px 0;
    color: white;
}
```

3.3 JavaScript

El tercer y último archivo contiene todo el código JavaScript, en el cual vamos a explicar minuciosa y detalladamente todas las líneas de código, ya que este archivo es el que hace que nuestra página web sea dinámica y permite al usuario interaccionar de una manera correcta y segura con todos los campos que se le presentan.

Procedemos con la explicación de los siguientes fragmentos de código del archivo JavaScript: "**cart.js**"

```
window.addEventListener("load",() =>{
    initVariables();
    initListeners();
    printing.disabled = true;
    cardPay.style.display = "none";
    cashPay.style.display = "none";
});
```

Lo primero que haremos dentro de nuestro archivo JavaScript será utilizar el objeto "**window**", el cual representa la ventana que contiene un documento DOM, seguido de un punto llamaremos al método ***addEventListener()*** que registra un evento a un objeto específico y permite dos argumentos:

- El primero de ellos será "**load**", este evento se activa cuando se ha cargado toda la página, incluidos todos los recursos dependientes, como hojas de estilo e imágenes.
- El segundo argumento es una función de flecha anónima y auto invocada en la que llamamos a otras funciones como son ***initVariables()*** y ***initiListeners()***, que como su propio nombre indica, inicializan las variables de nuestro archivo así como los escuchadores de eventos como el que hemos declarado anteriormente.

  Observamos tres líneas de código adicionales, que por ahora obviaremos con el fin de explicarlas posteriormente.

```
let cart = [];
let article;
let price;
let units
let add;
let allArticles;
let totalPrice;
let total;
let errorArticle;
let errorPrice;
let selectPay;
let cardPay;
let cashPay;
let cashDelivered;
let cardHolder;
let erase;
let printing;
let check;
let option;
let priceT = /^[+-]?\d+([.]\d+)?$/;
```

Una vez terminamos con la carga de la página, podemos bajarla unas cuantas líneas y en el inicio de nuestro archivo JavaScript, empezamos a declarar todas nuestras variables. Hacemos su declaración a través de la palabra reservada "*let" o "const"* en caso de que sea una constante que no vaya a ser modificada durante todo el código. También podríamos declarar las variables con "*var*", pero esto no es una buena práctica, ya que las variables pueden alterar sus valores de forma accidental y eso puede dar problemas de codificación y además ser difícil de depurar.

Para terminar con las variables, hacemos referencia a una declaración especial "PriceT", la cual es una *expresión regular*, que utilizaremos para validar el campo del formulario referente al precio del artículo, con el fin de que el usuario no pueda meter letras o símbolos, únicamente números enteros o con decimales separados por un ".".

```
function initVariables() {
    article = document.getElementById('articleName');
    price = document.getElementById('articlePrice');
    units = document.getElementById('units');
    add = document.getElementById('add');
    allArticles = document.getElementById('cartArticles');
    totalPrice = document.getElementById('totalPrice');
    errorArticle = document.getElementById('spanName');
    errorPrice = document.getElementById('spanArticlePrice');
    selectPay = document.getElementById('selectWayToPay');
    cardPay = document.getElementById('cardPay');
    cashPay = document.getElementById('cashPay');
    cashDelivered = document.getElementById('cashDelivered');
    totalAmount = document.getElementById('cashAmount');
    cardHolder = document.getElementById('cardHolder');
    check = document.getElementById('terms');
    erase = document.getElementById('reset');
    printing = document.getElementById('print');
};
```

El siguiente paso va a ser el de inicializar las variables que hemos declarado, si hacemos memoria, podemos recordar que esta función fue llamada en la carga de la página, esto se decidió así para tener el código ordenado y modularizado, por lo que su comprensión es mucho más clara y se puede modificar de una forma mucho más dinámica.

La sintaxis para inicializar las variables es la siguiente: “nombreVariable” = document. “método a utilizar”();

Primero cogemos nuestra variable, en este caso “article”, seguida de un “=”, acto seguido escribimos la palabra reservada document, que representa cualquier página web cargada en un navegador. Posteriormente le sigue un “.” y tras él, escribimos el método a utilizar, en nuestro caso “getElementById” y en el argumento introducimos entre comillas el “id” que tiene asignado en el html dicho elemento, para tener control sobre él en el DOM del html.

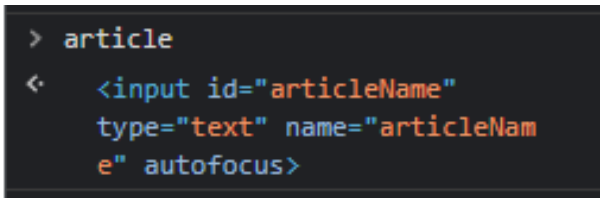


Imagen 1

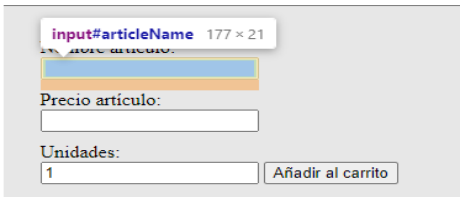


Imagen 2

Podemos cargar nuestra página web en un servidor local utilizando el IDE Visual Studio Code, por ejemplo, y si pulsamos la tecla f12 en nuestro navegador, podemos entrar a las herramientas de desarrollo y escribir el nombre de nuestra variable “article” en la consola. El sistema nos muestra la referencia de esa variable en el nuestro html, ya que la hemos referenciado con “id” que pusimos al principio, por ello es tan importante tener localizados nuestros elementos (*imagen 1)*.

Y lo segundo que podemos ver, es que, si pasamos el cursor por encima de ese texto html, automáticamente nos señala el elemento del formulario que estamos viendo en tiempo real con la página web cargada (*imagen 2).*

Repetiremos esta sintaxis con todas las variables que queramos inicializar para tener el control sobre ellas y poder dotar de funcionalidad a los diversos campos del formulario de nuestra página web.

```
function initListeners() {
    add.addEventListener('click',addProduct);
    erase.addEventListener('click',eraseForm);
    printing.addEventListener('click', printForm);
    check.addEventListener('change', ablePrint);
    selectPay.addEventListener('change', showPay);
};
```

La siguiente función que vamos a implantar, es la de iniciar todos los eventos de escucha o listeners, al igual que la anterior, está también fue llamada en la carga de la página.

La sintaxis es la siguiente: nombreVariable(inicializada).addEventListener(‘evento’, nombreFuncion);

En este caso, cogemos de ejemplo la variable “add” con ella declarada e inicializada, procedemos a invocar el método ***addEventListener()***, un escuchador que indica al navegador que esté atento a la interacción del usuario. Este método tiene dos argumentos:

- Primer argumento: Aquí tenemos que introducir entre comillas simples o dobles, el evento al que el navegador tiene que estar atento, este evento puede ser “**click**”, cuando el usuario haga un clic en un determinado lugar, “**change**”, cuando se produzca un cambio, por ejemplo, que se marque un checkbox o incluso un evento “**blur**”, que se active cuando el usuario se salga de una caja de texto.
- Segundo argumento: Hacemos referencia a una función, la cual cuando se produzca el evento seleccionado en el primer argumento, esta ejecute una serie de instrucciones modificando el comportamiento o prestando funcionalidad al formulario.

A partir de este punto, vamos a ir explicando cada función, las cuales se fueron realizando en orden, según los requerimientos del ejercicio.

```
function addProduct() {
    if (article.value == '' && price.value == ''){
        errorArticle.textContent = 'falta artículo';
        article.style.border = '2px solid red';
        errorPrice.textContent = 'falta precio';
        price.style.border = '2px solid red';
        article.focus();
    }else if (article.value == ''){
        errorArticle.textContent = 'falta artículo';
        article.style.border = '2px solid red';
        price.style.border = '2px solid black';
        article.focus();
        errorPrice.textContent  = '';
    }else if (price.value == ''){
        errorPrice.textContent = 'falta precio';
        price.style.border = '2px solid red';
        article.style.border = '2px solid black';
        price.focus();
        errorArticle.textContent = '';
```

```
    }else if (!priceT.test(price.value)){
        errorPrice.textContent = 'tipo de dato incorrecto';
        errorArticle.textContent = '';
        article.style.border = '2px solid black';
        price.style.border = '2px solid red';
        price.focus();
    }else{
        addArticle();
        addPrices();
        article.style.border = '2px solid black';
        price.style.border = '2px solid black';
        article.value = '';
        errorArticle.textContent = '';
        errorPrice.textContent  = '';
    }
}
```

La primera que nos encontramos es la función addProduct(), la cual verifica que no haya campos en blanco o que el usuario haya introducido algo incorrecto en los campos, como por ejemplo letras en el campo del precio. Si ocurre la situación en la que el usuario no digita un producto, el sistema no le dejará avanzar y además de poner el borde del campo en color rojo, aparecerá el mensaje "**falta artículo**" el cual hemos predefinido con el método "textContent" y hemos asignado a la variable que hace referencia al .

En el campo de precio tenemos tres situaciones:

- Si este queda vacío, entonces el campo tendrá el mismo comportamiento que el nombre del articulo mencionado anteriormente.
- En cambio, si rellena dicho campo con letras en vez de números, entonces es donde entra en juego una de las condiciones else-if, que valida precisamente el campo del precio a partir de la expresión regular que mencionamos anteriormente, impidiendo al usuario teclear algo que no sean números y advirtiendo con el mensaje "**tipo de dato incorrecto**" y actualizando el color del borde del campo de texto a rojo.

- La última situación es que el usuario haya digitado un artículo o un precio correcto, sabremos que es valido puesto que no saldrá ningún mensaje de error y además el borde la caja se repasará finamente con un color negro debido nuevamente a las propiedades ".style.border = "**2px solid black**". Además, cuando ambos campos estén correctamente cumplimentados, el usuario volverá a pulsar el botón "añadir al carrito", se añadirán los artículos y los precios que el usuario haya digitado, debido a la llamada de las funciones ***addArticle()*** y ***addPrices()***, se reseteará cualquier valor en el campo de precio, así como cualquier mensaje de error anterior, una vez que se complete con éxito la acción de añadir un artículo y un precio al carrito de compra.

En todos los casos con la función **focus**() hacemos que se ponga la atención en la caja de texto que tiene que rellenar el usuario.

```
function addArticle() {
    cart.push(article.value);
    showAllArticles();
}
```

La siguiente función en la que nos centramos es la de ***addArticle()***, la cual estaba llamada en la última condición "else" de la función ***addProduct()*** que hemos comentado unas líneas más arriba.

Su funcionamiento es el de añadir todos los artículos que el usuario a tecleado (article.value) al array "cart []" utilizando el método "**push()**" cuyo funcionamiento es añadir uno o más elementos al final del array y devolver nuevamente la longitud del mismo.

Finalmente llamamos a la función ***showAllArticles()***, para ir mostrando los artículos que se van guardando en el array, pero esto lo explicaremos más adelante.

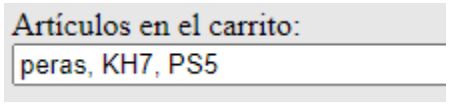
```
function addPrices() {
    let p = Number(price.value);
    let u = Number(units.value);
    let tp = Number(totalPrice.value);

    total = (p * u) + tp;
    showTotalPrice();
    price.value = '';
    price.style.border ='2px solid black';
    units.value = 1;
}
```

Ahora vamos a detenernos en una función importante, concretamente en ***addPrices()***, la cual hemos utilizado para ir sumando al importe final, los precios y unidades de cada artículo. Puesto que todos los campos de formulario son de tipo “text”, hemos tenido que convertir toda la información que un usuario digite para poder trabajar con ella y hacer las operaciones necesarias.

Podemos distinguir tres partes diferenciadas en esta función:

- Conversiones: Declaramos tres variables locales, las cuales van a estar igualadas a los valores que digite el usuario en las cajas de texto, pero no sin antes estar recogidas entre los argumentos del método Number(), lo que genera que se retornen como “number” y no como “text”.
- Operaciones: Con una simple operación matemática, multiplicamos el precio por las unidades y las sumamos al precio total, todo ello guardado en una variable global, llamada “total”.
- Visualización: Mostramos en el formulario el precio total, llamando a la función ***showTotalPrice()***, que explicaremos próximamente y formateamos el precio del artículo, a continuación seteamos nuevamente las unidades al valor de “1”, además de marcar en negro el borde de la caja de precio a modo de validación en caso de que hubiera habido un error anteriormente.

```
function showAllArticles() {
    allArticles.value = cart.join(', ');
}
```



Como comentábamos anteriormente, esta función había sido llamada en ***addArticle()***, su funcionamiento se basa en que el campo de texto que engloba todos los artículos que el usuario quiere comprar, se iguala a la variable "cart", que recordemos que es un array, seguida del método join(), con el que separamos todos los elementos de array utilizando una "**,**" como podemos ver en los argumentos, devolviéndonos una cadena con todos los elementos unidos.

```
function showTotalPrice() {
    totalPrice.value = total.toFixed(2);
}
```

El funcionamiento de ***showTotalPrice()***, no es más que el de mostrar el precio total de los artículos del carrito además de redondear dicha cantidad a dos decimales utilizando el método ***toFixed()*** introduciendo el número 2 entre sus argumentos. Este pequeño redondeo es necesario en caso de que el usuario introduzca números reales en el campo del precio de los artículos.

```
function showPay() {
    if(selectPay.value == 'T'){
        cardPay.style.display = 'block';
        cashPay.style.display = 'none';
        cardHolder.focus();
    }else if(selectPay.value == 'E'){
        cardPay.style.display = 'none';
        cashPay.style.display = 'block';
        cashDelivered.focus();
        if(totalPrice.value == 0){
            totalAmount.value = 0;
        }else{
            totalAmount.value = totalPrice.value;
        }
    }else{
        cardPay.style.display = 'none';
        cashPay.style.display = 'none';
    }
}
```

La siguiente función a analizar es ***showPay()***, la cual permite al usuario desplegar u ocultar las diferentes secciones del formulario según la forma de pago elegida. Nuevamente tenemos varios casos que procedemos a explicar:

- **Caso nº1**: El usuario podrá elegir pagar en tarjeta o en efectivo, por lo que elegirá una opción del selector. Ambas opciones desplegarán una serie de campos:
    - titular tarjeta, número tarjeta y CVV en caso de elegir la opción tarjeta.
    - importe efectivo y efectivo entregado en caso de escoger efectivo.

  Esto es posible, ya que las variables “cardPay” y “cashPay”, tienen su scope en las etiquetas , que engloban todos los campos mencionados anteriormente. Esto nos permite acceder tanto a los valores de las propiedades como a los textos si fuera el caso. Por lo tanto, utilizando el evento **change** el cual está ligado a esta función, podemos programar que cuando el usuario elija la opción “tarjeta” o “efectivo”, salgan sus respectivos campos, debido al uso de las propiedades .style.display =”**block**”;

- **Caso nº2:** Puede darse la situación de que el usuario se equivoqué al elegir el método de pago y quiera cambiar por lo que es importante que cada vez que el usuario escoja una de las opciones, la opción contraria se repliegue, para así poder navegar entre ambas, esto lo volvemos a conseguir a través de las propiedades style.display = "**none**";
- **Caso nº3:** En este supuesto, cuando el usuario elige la opción "efectivo", tenemos que mostrar el total del carrito, según los precios de los artículos que quiera comprar, por lo que igualamos el valor de ese campo al del precio total de carrito. Ahora bien, si el usuario elige la opción de efectivo antes de introducir ningún artículo y por lo tanto ningún precio, debemos introducir una nueva condición if-else que permita evaluar el precio total del carrito y si este se encuentra a 0, entonces seteamos el valor del campo del importe total también a cero, porque de lo contrario nos lo encontraríamos como "undefined".
- **Caso nº4:** Si en cualquier momento el usuario quiere cerrar estos desplegables deberá pinchar nuevamente en la opción por defecto "seleccione" y cualquiera de las dos secciones de formulario que esté desplegada se recogerá.

```
window.addEventListener("load",() =>{
    initVariables();
    initListeners();
    printing.disabled = true;
    cardPay.style.display = "none";
    cashPay.style.display = "none";
});
```

También podría pulsar el botón restablecer, que recarga la página según la función ***eraseForm()*** (que explicaremos más adelante) y ambas opciones se encuentran nuevamente recogidas, gracias a las instrucciones indicadas en la carga de la página.

```
function cashBack(){
 let cashReturn = Number(cashDelivered.value);

    if(cashReturn < totalPrice.value){
        return 'El efectivo entregado no es suficiente roñoso';
    }else {
        cashReturn -= totalPrice.value;

        return cashReturn.toFixed(2) + ' €';
    }
}
```

cashBack() es la función que hace referencia al cálculo de las vueltas que debe recibir el usuario cuando este paga en efectivo. Esto se traduce en una condición if-else que evalúa, que, si el efectivo entregado es menor que el precio total del carrito, se retornará el mensaje "*El efectivo entregado no es suficiente roñoso*".

El caso contrario retornará la resta del efectivo entregado frente al precio total, guardamos el resultado en la variable local *cashReturn* y además redondeamos nuevamente a dos decimales el cambio a entregar y concatenamos el símbolo de "€".

```
function ablePrint() {
    if(check.checked){
        printing.disabled = false;
    }else{
        printing.disabled = true;
    }
}
```

```
window.addEventListener("load",() =>{
    initVariables();
    initListeners();
    printing.disabled = true;
    cardPay.style.display = "none";
    cashPay.style.display = "none";
});
```

Imagen1

```
check.addEventListener('change', ablePrint);
```

Imagen2

Volvemos a hacer referencia a la carga de la página(imagen1) para la explicación de la función ***ablePrint()***, donde por defecto, tenemos el botón imprimir deshabilitado y volviendo a poner el foco en la función, utilizamos una condición if-else donde se nos presentan dos posibles casos:

- Una primera condición en la que utilizamos la variable "check", la cual está referenciada directamente al "id" correspondiente al checkbox de los términos y condiciones (imagen2), por lo que utilizando la propiedad .checked, obtenemos "**true**", en caso de que el checkbox este marcado, por lo tanto, el botón pasa a estar habilitado a través de la propiedad. disabled = **false**;
- Una segunda condición que nos dice que mientras el checkbox no esté marcado, el botón "imprimir" estará deshabilitado, impidiendo al usuario finalizar con la compra.

```
function printForm(){
    if(selectPay.value == 'T'){
        option = confirm('Los artículos de mi carrito son: ' +
allArticles.value +"\n" +
                    'El precio total es: ' + totalPrice.value + ' €' + '\n'
+
                    'Forma de pago: Tarjeta' + "\n" +
                    '¿Estás seguro de comprar estos artículos?');
        if(option == true){
            print();
        }
    }else if(selectPay.value == 'E'){
        option = confirm('Los artículos de mi carrito son: ' +
allArticles.value +"\n" +
                    'El precio total es: ' + totalPrice.value + ' €' + '\n'
+
                    'Forma de pago: Efectivo'  + "\n" +
                    'Efectivo a devolver: ' + cashBack() + '\n' +
                    '¿Estás seguro de comprar estos artículos?');
        if(option == true){
            print();
         }
    }else{
        alert('Seleccione una forma de pago');
    }
```

La función printForm() nos permite visualizar una ventana modal con todos los datos de la compra o por el contrario mostrar una alerta en caso de que el usuario no haya escogido una forma de pago. Vamos a centrarnos en explicar cada caso detalladamente:

- **1º caso:** La condición evalúa si el usuario a elegido el método de pago tarjeta ("**T**"), en este caso, con un método **confirm*()***, mostramos al usuario todos los productos, el importe a pagar y la forma de pago escogida, dándole la opción de confirmar. Todo ello lo hemos guardado en una variable global llamada option, que cogerá el valor de True o False, en cuanto el usuario pulse el botón aceptar o cancelar que le aparece en pantalla. Si la variable "option", obtiene en valor de True, entonces entrará en juego el siguiente if anidado, que mandará la opción de imprimir la pantalla con todos los datos introducidos.
- **2º caso:** Este segundo caso, es muy parecido al anterior, con la diferencia de que al tratarse de la forma de pago en efectivo, se añaden pequeños cambios a mostrar, ya que el usuario ha tenido que introducir anteriormente el importe de efectivo que va a entregar, por lo que el sistema mostrará un campo adicional que será el efectivo a devolver, (***+ cashBack***) o el mensaje que indicará al usuario que no ha introducido efectivo suficiente para saldar la cuenta, además de imprimir la página si el usuario acepta.
- **3º caso:** En caso de que el usuario no haya elegido una forma de pago, utilizaremos el método **alert()** con el que se mostrará un diálogo de alerta con el mensaje "Seleccione una forma de pago" que se mantendrá hasta que el usuario cierre dicho cuadro de diálogo.

```
function eraseForm() {
    location.reload();
}
```

Terminamos con la última función del código JavaScript, la cual va ligada al botón “restablecer” y cuyo funcionamiento es el de recargar la página restableciendo todos los campos a través del método **location.reload()**, que recarga la URL actual.

4 Conclusiones

Este trabajo en conjunto, nos ha gustado mucho, puesto que hemos podido aprender un nuevo e interesante lenguaje de programación como es JavaScript, además de reforzar todos los conocimientos que hemos ido viendo durante el curso y algunos otros que ya sabíamos pero que ha sido importante refrescar mientras realizábamos esta práctica.

Además, nos hemos enfrentado a varios retos mientras avanzando en nuestro proyecto, lo que también nos aporta una experiencia de cara a posibles problemas en un futuro al realizar este tipo de acciones. Ha sido interesante que cada integrante del grupo, aportara sus propias pruebas y argumentos de todos y cada uno de los problemas que se iban presentado durante la realización de la activad con el fin de comparar y poder debatir acerca de cuál sería la forma más adecuada de realizar ciertas acciones.

Algunas de las decisiones más importantes que hemos tomado en conjunto, han sido acerca de la presentación del proyecto, utilizando en todo momento buenas prácticas y que todo quedará de la forma más clara y ordenada. También hemos debatido acerca de cómo implementar ciertas funciones y métodos, así como la paleta de colores que hemos utilizado para dar estilos a nuestro trabajo. Aunque algunas decisiones tuvieran más peso que otras al final ha sido fundamental ponernos de acuerdo, escuchando a todas las partes para sacar adelante un buen proyecto.

Al final esta metodología híbrida nos ayuda a trabajar por cuenta propia, pero sin olvidar lo importante que resulta la puesta en común, para que esta sea enriquecedora y que además tengamos que pensar como representar en este documento formal de manera clara y concisa todo el trabajo que agrupamos entre los tres miembros del equipo.