Flujo (Parte I)

Diagramas de Flujo y Pseudocódigo

Competencias

- Conocer los principios básicos de un pseudocódigo.
- Memorizar la simbología básica de los diagramas de flujo.
- Construir diagramas de flujo y pseudocódigo.

Introducción

Pese a la creencia de que un algoritmo es necesariamente un programa o es de exclusiva pertenencia de un código, un algoritmo no tiene relación directa con programación, sino que es, en palabras simples, un conjunto de pasos necesario para resolver un problema o lograr un objetivo. Esto no significa que no están relacionados, sino que son los programas que tienen su base en los algoritmos, por ende, antes de comenzar a escribir código es necesario comprender qué es un algoritmo.

Como un primer acercamiento a lo que son los algoritmos y su entendimiento, se abordan los diagramas de flujo, herramienta que nos permite representar un flujo de trabajo o proceso, es por esta misma razón que es conveniente iniciar en el mundo de la programación con **diagramas de flujo** y **pseudocódigo**, para poder estimular el pensamiento algorítmico y ser capaz de representar tus ideas de manera gráfica, para también lograr comunicarse con personas que solo conocen de diagramas.

De igual forma, es posible transmitir el proceso de manera menos gráfica escribiendo las instrucciones de forma secuencial, limitándose escribir solo la información necesaria, este método se conoce como **pseudocódigo**.

Pseudocódigo

Un pseudocódigo es una forma informal de escribir las operaciones de un algoritmo, dando enfoque en los datos más relevantes de dichas instrucciones acercándose, de esta manera, a lo que sería un código escrito en algún lenguaje en específico.

Dado que no existe una sintaxis estándar o formal para escribir pseudocódigo (la que más se acerca a ésta es la matemática), sólo nos limitaremos a escribir las instrucciones en español, reflejando lo mejor posible, acorde a los conocimientos de cada uno, al lenguaje JavaScript. Cabe recalcar que la intención principal del pseudocódigo es que sea interpretado por personas y no por máquinas, por lo que debe ser cercano al lenguaje humano.

Por ejemplo, un algoritmo, para pasar la lista de un curso en pseudocódigo, según un nivel más “humano”, sigue la siente forma:

```
Por cada estudiante en el listado de estudiantes:
    Si el estudiante está presente entonces:
        Marcar como presente
    Sino:
        Marcar como ausente
```

Representación de Diagramas de Flujo

Es posible definir un algoritmo para toda clase de tareas triviales, como preparar una comida, realizar un viaje, escribir un documento. En este mismo contexto, se utiliza el diagrama de flujo para poder transmitir y documentar actividades o flujos de datos a otras personas que no necesariamente conocen un lenguaje de programación, pero sí pueden entender de manera gráfica la secuencia de pasos a seguir para realizar una tarea.

	Línea de flujo, representa la dirección en que se mueve el proceso.
	Inicio/Terminal, indica el punto de inicio o término de un proceso.
	Proceso, representa una operación (acción), un cambio de valor, una lectura, una escritura, etc. por lo mismo se describe con un verbo.
	Decisión, representa una toma de decisión o condicional en donde existe más de un camino o flujo de datos y se debe escoger uno en base a una pregunta o condición. El resultado de esta operación debe ser si/no o verdadero/falso.

Tabla 1. Simbología en la Representación de Diagramas de Flujo.

Apliquemos esto en un ejemplo básico, práctico y bastante cotidiano, es posible asumir que todo quién vive en Santiago ha utilizado alguna vez en su vida el servicio de trenes del Metro, por ende, sabe como llegar de una estación a otra. Para este ejemplo solo utilizaremos el caso básico en que una persona va una de una estación de metro a otra en la misma línea, es decir, sin realizar ninguna combinación o transbordo.

Luego, los pasos a seguir son:

1. El usuario aborda el tren.
2. El tren avanza de estación en estación (1 estación a la vez).
3. El pasajero verifica si la estación a la que ha llegado el tren corresponde a la de su destino.
4. De ser así, el pasajero desciende del tren y finaliza su recorrido.
5. En caso contrario se mantiene en el tren y este avanza una estación nuevamente (vuelve al paso 2).

Debido a que tanto texto con condicionales incluidas complica la comprensión del proceso se crea el diagrama:

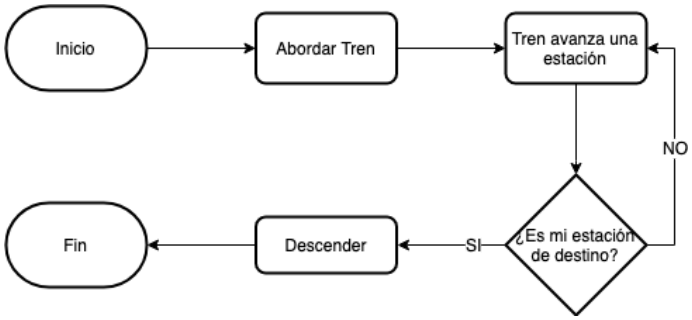

Imagen 1. Diagrama de Flujo - Ejemplo básico.

Este diagrama es bastante básico y puede ser descrito con mayor o menor detalle, por ejemplo, incluyendo más detalles:

1. Tras pagar su pasaje, el usuario se dirige al andén a la espera del tren.
2. Si el andén se encuentra en el piso inferior este debe descender, en caso contrario ascender (Rombo de decisión)
3. El pasajero espera en el andén hasta que llegue un tren.
4. Cuando llega un tren el pasajero debe verificar si corresponde a la ruta que él necesita (Roja/Verde para el caso de las líneas 2, 4 y 5).
5. Si corresponde, abordar el tren, en caso contrario, debe esperar el siguiente (Volver al paso 3).
6. Una vez aborda, el tren se mueve de 1 o 2 estaciones según sea el caso (existen rutas de colores o no).
7. Cuando el tren se detiene el pasajero debe verificar si la estación a la que ha llegado es su estación de destino.
8. Si es la estación de destino, el pasajero desciende, en caso contrario, se mantiene en el tren hasta la siguiente detención.
9. Una vez llega a su destino el pasajero abandona las dependencias de Metro.

El nivel de detalle y especificación de cada diagrama queda a juicio de quién lo diseña y de los requerimientos que debe cumplir.

Volviendo al ejemplo básico, una opción de pseudocódigo es:

```
abordar tren
estación actual = ‘Pajaritos’
estación destino = ‘Salvador’mientras la estación actual sea diferente a la
estación de destino entonces:
        avanzar tren   // estación actual avanza una estación
descender tren
```

Que en código JavaScript quedaría algo así:

```
abordarTren();
var estacionActual = ‘Pajaritos’;
var estacionDestino = ‘Salvador’;
while estacionActual != estacionDestino:
    estacionActual = avanzarTren();
descenderTren();
```

A medida que avancemos en el documento abordaremos el significado de cada una de las líneas anteriores.

Introducción a JavaScript

Competencias

- Entender qué es Javascript, su lógica, cómo funciona y cómo aplicarlo en la solución de problemas prácticos.
- Conocer un poco de su historia y relevancia en la programación mundial.
- Entender la consola de desarrollo en navegadores.
- Comprender por qué utilizar Javascript y no otro lenguaje.

Introducción

Javascript es un lenguaje de programación que se utiliza principalmente para el desarrollo de páginas web dinámicas. **¿Qué quiere decir dinámicas?**, que tienen movimiento, interacción con el usuario, son páginas web con efectos de textos, que al hacer click en algún botón este realiza alguna acción específica, como por ejemplo: guardar, eliminar, mostrar un mensaje, realizar un cálculo o enviar los datos que queramos.

Técnicamente Javascript es un lenguaje de programación el cual se le llama **“Lenguaje Interpretado”**. ¿Y esto qué quiere decir?, que no es necesario compilar nuestra aplicación en algún ejecutable para que este funcione, sino que basta solamente probar nuestro código escrito en Javascript en cualquier navegador web sin ningún tipo de procesos intermedios, a diferencia de otros lenguajes de programación.

Dado que comparten una palabra al inicio de su nombre, muchos piensan que Javascript y el lenguaje de programación Java tienen algún tipo de relación, o que Javascript nació o es un subconjunto de Java. Pero no es así, ambos lenguajes nacieron de empresas distintas y para distintos propósitos.

Para entender de qué manera funciona Javascript hay que ver con qué otros actores se encuentra relacionados. Esencialmente dentro del desarrollo web Front-End, Javascript se encuentra estrechamente relacionado con HTML y CSS.

En esta Imagen podemos ver de qué forma podríamos definir estos lenguajes:

Imagen 2. Algunos Lenguajes de Programación.

- **HTML** se encarga del contenido y de la **estructura** que le vamos a mostrar al usuario.
- **CSS** se encarga de la parte netamente visual, **estilos**, la estética y apariencia del sitio.
- Mientras que **JAVASCRIPT**, se encarga de la **funcionalidad**, es decir, que el usuario tenga una interacción más significativa con el sitio que estamos desarrollando.

Para que entendamos mejor esto que acabamos de explicar veremos un sencillo ejemplo en un formulario de registro de datos, donde se validará si los campos están o no llenados:

- **Sólo con HTML (Estructura)**

Imagen 3. Formulario de Registro con HTML

```html
<html>
   <head>
           <meta charset="utf-8"/>
           <meta name="description" content="Formulario de validación">
           <title>Formulario de validación</title>
           <link rel="stylesheet" href="estilo.css" type="text/css"/>
           <script type="text/javascript" src="validador.js"></script>
   </head>
   <body>
       <h2>FORMULARIO DE REGISTRO</h2>
       <form onsubmit="validacion();">
           <p>Nombre de usuario</p>
           <input type="text" class="field" id="username" />
           <p>Contraseña</p>
           <input type="password" class="field" id="pass" />
            <p>Email</p>
           <input type="text" class="field" id="email" />
           <p class="center-content">
               <input type="submit" class="btn" id="submit" value="Crear" />
           </p>
       </form>
   </body>
</html>
```

- **Formulario con HTML (Estructura) y CSS (Estilos)**

Imagen 4. Formulario de Registro con HTML y CSS.

```css
form{
   background-color: rgb(187, 186, 186);
   border-radius: 10px;
   color: rgb(255, 255, 255);
   font-size: 14px;
   padding: 20px;
   margin: 0 auto;
   width: 300px;
}
input, textarea{
   border: 20;
   outline: none;

   width: 280px;
}
.field{
   border: solid 1px #ccc;
   padding: 10px;
}
.field:focus{
   border-color:  rgb(124, 193, 250);
}

.center-content{
   text-align: center;
}
h2{
   text-align: center;
   color: rgb(124, 193, 250);
}
```

- **Formulario con HTML (Estructura), CSS (Estilos) y JAVASCRIPT (Funcionalidad)**

Imagen 5. Formulario de Registro con HTML, CSS y Javascript.

```
function validacion(){
   var user = document.getElementById("username").value;
   var pass = document.getElementById("pass").value;
   var email = document.getElementById("email").value;

   if (user == ""){
       alert("Por favor ingrese un usuario");

   }

   if (pass == ""){
       alert("Por favor ingrese su contraseña");

   }

   if (email == ""){
       alert("Por favor ingrese su email");

   }

   if ((user != "") && (pass != "") && (email != "")){
       alert("Gracias por registrarte!")

   }
}
```

En este básico y muy sencillo ejemplo, pudimos ver de qué manera Javascript está estrechamente relacionado con HTML y CSS, y de qué manera da funcionalidad a las estructura y estilos de un código.

El método que utilizamos en este ejemplo y que utilizaremos más adelante también es el "getElementById", que sencillamente sirve para referenciar un elemento (tag o etiqueta) por su atributo id, que se supone **debe ser único**. El método devuelve la etiqueta cuyo atributo id coincide con el parámetro del método. Este es uno de los métodos más comunes en el DOM de HTML y se usa casi cada vez que desea manipular u obtener información de un elemento de su documento.

Características Generales de Javascript

1. **Liviano:** Es un lenguaje realmente liviano, su carga no es mucha y permite una correcta ejecución sin necesidad de tener grandes características de hardware.
2. **Multiplataforma:** Javascript depende más del navegador que del sistema operativo sobre el cual se está ejecutando. Es decir, una aplicación hecha en Javascript va poder ejecutarse de igual manera en Linux, Windows o Mac OS X sin ningún problema.
3. **Multiparadigma:** Es un lenguaje que cumple con múltiples paradigmas de programación (formas de programar), es orientado a objetos, orientado a eventos, de carácter imperativo, entre otros.
4. **Interpretado:** Al ser un lenguaje interpretado no necesitamos compilar nuestro código a diferencia de otros lenguajes para ejecutarlo o ver los cambios realizados, basta solamente con modificar o actualizar nuestro archivo y podremos ver el resultado de inmediato en el navegador.

Hasta aquí probablemente para aprender html y css has sentido que ha sido como escalar una montaña, bueno, no lo es todo, ahora nos disponemos a escalar esta montaña llamada Javascript. No te asustes, juntos aprenderemos y te darás cuenta que con práctica y perseverancia daremos con nuestros objetivos.

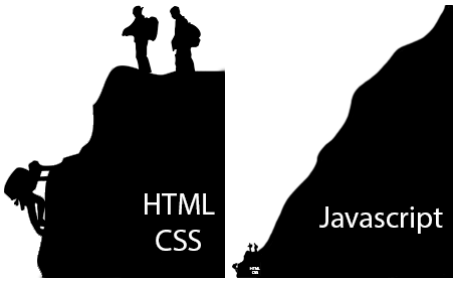

Imagen 6. HTML-CSS vs JavaScript

Sintaxis del Lenguaje

La sintaxis de un lenguaje de programación se refiere al conjunto de reglas que se deben seguir al momento de escribir el código. Esto quiere decir que hay formas de escribir un lenguaje de programación y no es llegar y escribir cosas a tontas y a locas.

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como de Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

- **No se toman en cuenta los espacios en blanco y las nuevas líneas:** El intérprete de JavaScript ignora cualquier espacio en blanco que sobre, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)
- **Distinguen las mayúsculas y minúsculas**: En JavaScript si se intercambian mayúsculas y minúsculas en el código el script no funciona. Puedo tener dos variables llamas: numero y otra llamada Numero. Ambos son distintas.
- **No se define el tipo de las variables:** al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
- **No es necesario terminar cada sentencia con punto y coma (;):** Aunque JavaScript no obliga a hacerlo, es conveniente y se **recomienda** de terminar cada sentencia con el carácter del punto y coma (;).
- **Se pueden incluir comentarios:** los comentarios se utilizan para añadir información en el código. Aunque el contenido de los comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.

Breve Historia de JavaScript

En el año 1995 fue creado Javascript, la razón de que este lenguaje fuera creado es que por aquel entonces existían dos empresas basadas en la web que estaban compitiendo. Estas empresas eran Netscape y Microsoft, cada una tenía un navegador y cada empresa su deseo era que este fuera el más usado para navegar por internet. En este deseo cada empresa estaba agregando a sus navegadores mayores funcionalidades, en donde esto permitía a los desarrolladores crear sitios web más atractivos a los usuarios, que no solamente estos puedan ver información sino interactuar y mucho más.

Con esta idea Netscape creo Javascript, la idea de crear un lenguaje de programación que funcionara dentro del navegador era que los desarrolladores pudieran hacer que los sitios o páginas web pudieran hacer mucho más. Por ejemplo, que los usuarios pudieran ejecutar algún mini juego dentro del navegador, que los usuarios pudieran cargar más rápidamente las páginas, en resumen, esta fue la idea por la que Javascript se creó, para extender la funcionalidad de que por aquel entonces las páginas podían solamente mostrar información.

La primera versión de JavaScript fue todo un éxito y Netscape Navigator 3.0 ya incorporaba la siguiente versión del lenguaje, la versión 1.1. Por otro lado, la empresa Microsoft lanzó JScript con su navegador Internet Explorer 3. Pero JScript era nada más ni nada menos que una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.

Dado todo esto comenzó a surgir la idea de que cada navegador podría tener o usar su propio lenguaje, lo que esto significaba más un problema que un aporte. Es por esto que Netscape quiso establecer Javascript como un estándar. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA European Computer Manufacturers Association).

Bajo ésta, en la actualidad todos los navegadores utilizan Javascript para añadir interactividad a sus páginas web, hoy en día todos los navegadores tiene que regirse a una norma de como leer o utilizar Javascript. Es por eso que cuando visitamos una página web o cuando la desarrollamos esta se ve exactamente igual en todos los navegadores, o lo más parecido posible a la original.

Entendiendo la Consola de Desarrollo en Navegadores

La consola de desarrollo de Javascript es una herramienta maravillosa a la hora de analizar nuestro proyecto cuando estamos programando. Esta nos muestra mensajes de información, error o alerta. Además incluye inspectores o verdaderos depuradores de código que nos permiten identificar para así corregir errores.

También nos permite interactuar con la página, ejecutando expresiones o comandos de JavaScript.

Para acceder a la consola desde Google Chrome y Firefox es simplemente presionando las teclas ***Control + Mayus + J***, desde Mac ***cmd + alt + J***. En tanto desde Internet Explorer desde la tecla f12.

La consola, como podemos ver en la siguiente imagen, muestra varias pestañas en las que se pueden encontrar información de las peticiones HTTP (los elementos de la página que se cargan de internet), los errores y análisis del CSS, JavaScript, errores y advertencias de seguridad y los Logs (mensajes).

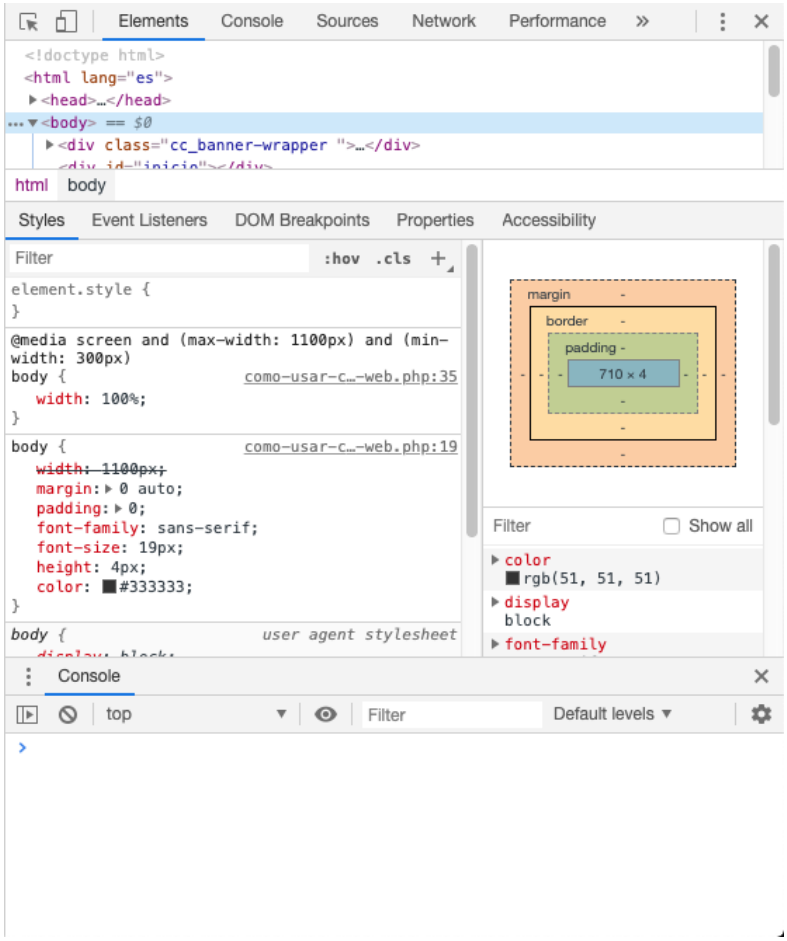

Imagen 7. Consola en Navegadores Web.

A continuación realizaremos un ejemplo muy breve y básico para que podamos ver cómo la consola ejecuta el código Javascript de manera automática.

Abrimos la consola con ***Control + Mayus + J*** ó ***cmd + alt + J*** desde Mac y se nos abrirá de la siguiente manera:

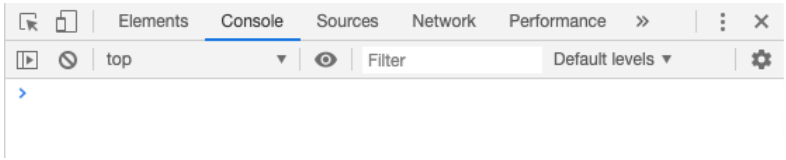

Imagen 8. Consola en Navegadores Web desde Mac.

Lo primero que haremos de manera extremadamente básica es una sumatoria matemática, para que puedas observar que la consola trabaja automáticamente bien.

Haz la siguiente operación y obtendrás el resultado:

- **10 + 15**

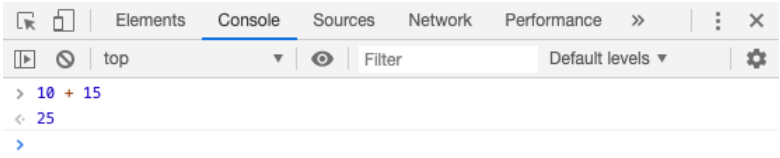

Imagen 9. Sumatoria en Consola.

Ahora, esta misma operación la haremos utilizando un poco de Javascript. Sino entiendes nada de esto, tranquilo/a más adelante veremos detalladamente cada detalle que en este ejemplo probablemente no entiendes.

Se declara dos variables, num1 y num2. cada variable almacenará un valor y luego las sumaremos.

- **var num1 = 10;**
- **var num2 = 15;**

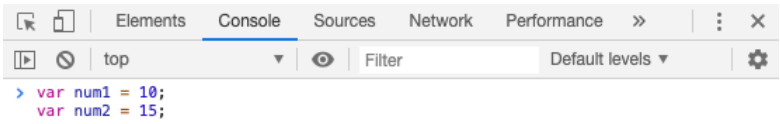

Imagen 10. Sumatoria en Consola con JavaScript.

Ahora, agregamos en la siguiente línea para obtener el resultado (para hacer un salto de línea, sin ejecutar, lo hacemos presionando las teclas **Shift + Enter**).

var num1 = 10; var num2 = 15;

- **console.log(num1 + num2);**

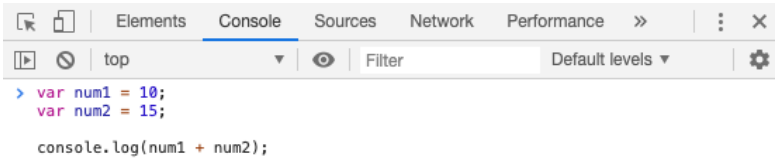

Imagen 11. Fórmula de sumatoria en Consola con JavaScript.

Damos enter y tendremos el resultado que es **25**.

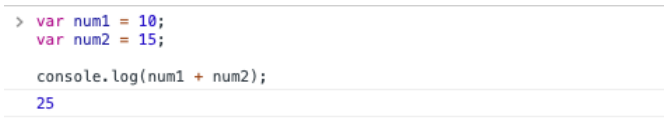

Imagen 12. Resultado de sumatoria en Consola con JavaScript.

Ahora haremos una tabla con datos, escribe el siguiente código en la consola:

```
console.table({Nombre : “Arturo”, Apellido : “Vidal”, Club : “Barcelona”});
```

Damos **enter** y obtendremos nuestra tabla con datos:

Imagen 13. Código para hacer una Tabla en Consola.

Por último, en este último ejemplo, agregaremos el siguiente código:

```
alert (“Hola! esto es Javascript en {desafio} latam_”);
```

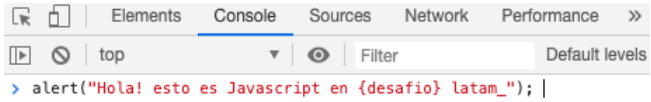

Imagen 14. Código en Consola.

Damos enter y obtendremos lo siguiente:

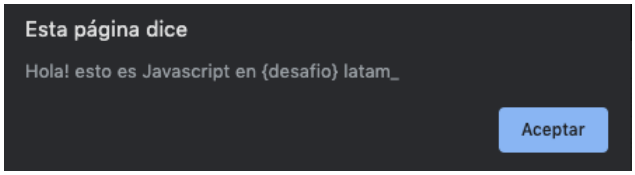

Imagen 15. Código en consola con JavaScript.

Por si no lo notaste, ¡Ya programaste en JavaScript! Ahora en los siguiente capítulos veremos detalladamente el lenguaje.

¿Porqué utilizar JavaScript?

Javascript se ha transformado en el lenguaje de programación del lado del cliente más popular y más utilizado por los desarrolladores Front-End. No porque este lenguaje sea la moda simplemente, sino que porque durante el tiempo ha demostrado ser un lenguaje sólido, liviano, simple y muy funcional.

La programación es la manipulación de datos, y JavaScript lo hace perfecto. JavaScript nos permitirá manipular el DOM (Document Object Model, "*Es la representación de la página HTML*"), obtener datos de sus elementos, modificarlos, animarlos, condicionarlos y procesarlos, guardarlos para ocuparlos más tarde, mandarlos o pedirlos al servidor, casi todo lo que se nos ocurra, podemos hacerlo con JavaScript.

Existen los llamados y populares frameworks, que son librerías de código, estructuras ya personalizadas y mejoradas en base a un lenguaje. La principal razón por la que los frameworks de JavaScript se han popularizado en los últimos tiempos es la experiencia web interactiva que nos ofrecen. Teniendo una base sólida de JavaScript tenemos la oportunidad de aprender y manejar cualquiera de los Frameworks de JavaScript y así desarrollar aplicaciones más robustas.

Algunos de los Frameworks de JavaScript más conocidos son:

Imagen 16. Frameworks de JavaScript.

¿Cómo incluir JavaScript en un archivo HTML?

La integración de JavaScript y HTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web.

1. Entre las etiquetas `<script>` dentro del código HTML.
2. Dentro del mismo DOM, como por ejemplo usando el atributo 'onClick="miFuncion();"'.
3. Entre las etiquetas `<script>`, pero esta vez usando el atributo "src" para darle la ruta del archivo .js que se utilizará.

Y puesto que uno los punto más importantes que todo programador debe mantener en su cabeza es la creación de un código limpio, ordenado y simple de entender. En base a esto nosotros utilizaremos la tercera forma de incluir JavaScript a HTML, y es incluirlo desde un archivo externo a nuestro archivo HTML.

Para comenzar creemos una carpeta en nuestro computador con el nombre de "Iniciar JavaScript en HTML"

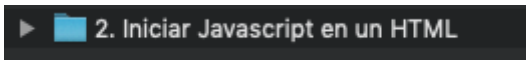

Imagen 17. Ejemplo de carpeta.

Luego con nuestro editor, en este caso utilizaremos Visual Studio Code, se abre la carpeta creada:

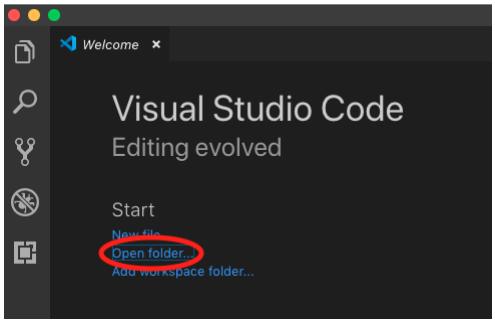

Imagen 18. Carpeta abierta con Editor Visual Studio Code.

Ahora crearemos nuestro archivo HTML con el siguiente código el cual guardaremos como "index.html"

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Mi primer Javascript - {desafío} latam_</title>
</head>
<body>
   <h1>Este es un documento HTML con Javascript</h1>
</body>
</html>
```

Ahora se creará un archivo JavaScript. Le pondremos como nombre “javascript.js”; los archivos Javascript son con la extensión **“.js”**, entonces creamos el archivo y el proyecto quedará de la siguiente manera:

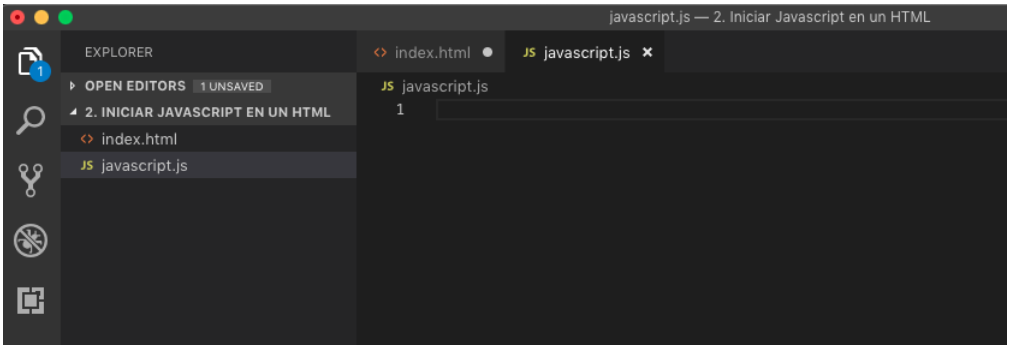

Imagen 19. Visualización de Proyecto.

Ya creado nuestro archivo “javascript.js” agregaremos el siguiente código

```
alert("Javascript está funcionando!");
```

Guardamos, y si abres el archivo “index.html” no notaras ningun cambio y verás solamente esto:

Este es un documento HTML con Javascript

Imagen 20. Visualización de resultado.

Y es porque no hemos hecho lo más importante, que es incluir nuestro archivo JavaScript. Para incluirlo agregamos la siguiente línea de código dentro de la etiqueta de nuestro HTML, pero lo agregamos antes de cerrar el , al final. Esto es porque de esta manera se ejecute primero todo nuestro sitio y luego nuestros scripts.

```
<script type="text/javascript" src="javascript.js"></script>
```

Este código básicamente declara que tipo de archivo estamos incluyendo y en el src definimos la ruta donde se encuentra nuestro archivo JavaScript. Es igual a cuando incluimos un archivo CSS.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,    initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Mi primer Javascript - {desafío} latam_</title>
</head>
<body>
   <h1>Este es un documento HTML con Javascript</h1>

   <script type="text/javascript" src="javascript.js"></script>
</body>
</html>
```

Guardamos y abrimos nuestro index.html y nos aparecerá lo siguiente:

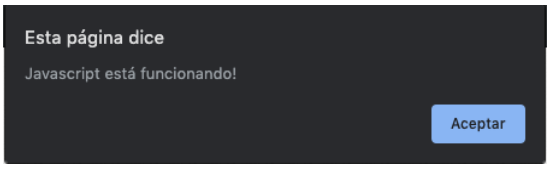

Imagen 21. Proyecto con Javascript.

Si aparece este mensaje es porque lo has hecho excelente!. No borres este proyecto, ya que, lo utilizaremos para hacer otros ejemplos.

Ya aprendimos lo básico para comenzar a programar en JavaScript, que es incluirlo en nuestras páginas en HTML.

Una de las ventajas de incluirlo en un archivo externo es que tenemos un código más limpio y más eficiente. Además este mismo archivo .JS lo podemos utilizar y llamar en diferentes archivos .HTML. sin necesidad de agregar el mismo código en todas nuestras páginas.

Tipos de datos en JavaScript

Competencias

- Memorizar cuáles son los tipos de datos y sus valores.
- Comprender la importancia de los tipos de datos.
- Capacidad de aplicar correctamente los tipos de datos a distintos tipos de valores.

Introducción

Como lo mencionamos en el capítulo anterior, la programación es la manipulación de datos. Frente a esto es de suma importancia conocer los tipos de datos que estaremos manipulando con este lenguaje de programación llamado JavaScript.

Por ejemplo, cuando estamos contratando un plan de teléfono, el ejecutivo nos solicita nuestros datos personales: Rut, Edad, Fecha de Nacimiento, Dirección, Teléfono y Correo electrónico. Aquí tenemos diferentes tipos de datos, números, fecha, mail, texto, etc. con los que el ejecutivo puede manipular como él quiera para elaborar un contrato.

De esta forma nosotros como programadores tenemos que conocer el tipo de dato que estaremos manipulando para ver de qué manera lo podemos hacer y trabajar de una manera más óptima.

JavaScript tiene definido los tipos de datos que podemos manipular, y los llama como tipos de datos primitivos. ¿A qué se refiere al decir datos primitivos? Se refiere a un tipo de dato que representa un solo valor y es inmutable (que no se puede modificar).

Tipos de Datos

En JavaScript tenemos 5 tipos de valores:

1. **Tipo Boolean** = Boolean representa una entidad lógica y puede tener dos valores: true o false.
2. **Tipo Number** = Con un valor Numérico.
3. **Tipo String** = Una cadena de Texto.
4. **Tipo Undefined** = Una variable a la cual no se le haya asignado valor tiene entonces el valor *undefined*.
5. **Tipo Null** = Una variable que se le asigna y representa un valor nulo o vacío.

Para hacerlo de manera práctica, ocuparemos nuestro editor de código, Visual Studio Code, ocupando el operador **"typeof"** que nos devolverá un string (texto) indicándonos a qué tipo de valor pertenece el dato que almacenaremos en una variable.

Iremos en orden, **utilizaremos el mismo proyecto que hicimos anteriormente**, en este caso abriremos el archivo que creamos y llamamos "javascript.js", donde agregaremos el siguiente código:

```
var verdadero = true;
// declaramos la variable "verdadero" con un valor Boolean.

alert(typeof verdadero);
// Este mensaje de alerta nos menciona el tipo de dato que es
```

(Cuando queremos comentar en JavaScript lo hacemos con // y luego el comentario)

Guardamos y ejecutamos nuestro "index.html" y nos aparecerá los siguiente:

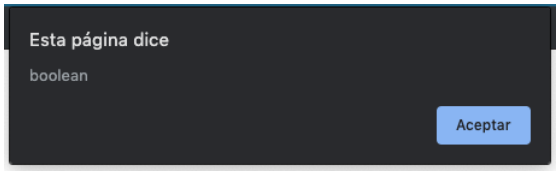

Imagen 22. Resultado ejecución de código utilizando el tipo de dato "boolean".

¿Qué fue lo que hicimos?

- Declaramos una variable la cual pusimos por nombre “verdadero”. *(Siguiente capítulo veremos las variables)* y a esta le dimos un valor, el cual fue “true”, y es un tipo de dato Boolean.
- Luego, ocupamos el operador **“typeof”** el cual simplemente nos devuelve un texto “string” con el tipo de dato que es nuestra variable, seguido del nombre de la variable. Todo esto dentro de la instrucción **alert()** para que nos muestre el tipo de dato en un mensaje o cuadro de alerta e el navegador.

Sigamos, ahora continuemos haciendo lo mismo pero con el siguiente tipo de dato primitivo.

Borremos el código que ingresamos en el archivo “javascript.js” y agreguemos el siguiente:

```
var numero = 10;
// declaramos la variable "numero" con un valor numérico.

alert(typeof numero);
// Este mensaje de alerta nos menciona el tipo de dato que es
```

Guardamos y abrimos el archivo “index.html” en nuestro navegador, y veremos lo siguiente:

Imagen 23. Resultado ejecución de código utilizando el tipo de dato “number”.

Este mensaje nos muestra el tipo de dato que es nuestra variable llamada **numero**.

```
var texto = "Hola Desafío Latam";
// declaramos la variable "texto" con un valor de texto.

alert(typeof texto);
// Este mensaje de alerta nos menciona el tipo de dato que es
```

Guardamos y abrimos el archivo “index.html” en nuestro navegador, y veremos lo siguiente:

Imagen 24. Resultado ejecución de código utilizando el tipo de dato “string”.

Este mensaje nos muestra que la variable que declaramos es de valor tipo **string**.

Repetimos los mismos pasos y agregamos el siguiente código en nuestro “javascript.js”:

```
var noExisto;
// Se declara variable sin ningún valor.

alert(typeof noExisto);
// Este mensaje de alerta nos menciona el tipo de dato que es.
```

Guardamos y abrimos el archivo “index.html” en nuestro navegador, y veremos lo siguiente:

Imagen 25. Resultado ejecución de código utilizando el tipo de dato “undefined”

Este mensaje nos muestra que el valor de la variable que declaramos **no está definido**.

Borremos el código que ingresamos en el archivo “javascript.js” y agreguemos el siguiente:

```
var vacia = null;
// Se declara variable con un valor de vacío.

alert(typeof vacia);
// Este mensaje de alerta nos menciona el tipo de dato que es.
```

Guardamos y abrimos el archivo “index.html” en nuestro navegador, y veremos lo siguiente:

Imagen 26. Resultado ejecución de código utilizando el tipo de dato “object”

Este mensaje nos muestra que la variable que declaramos es de valor tipo null como “object” ya que está declarada pero con un valor vacío *(a partir de ES6, también existe el tipo Symbol, que básicamente nos permite almacenar valores como number o string de manera inmutable)*.

En ejemplos ya más prácticos, en nuestro trabajo como desarrolladores vamos a ocupar tipos de datos **“string”** para manejar textos en nuestros elementos, ya sea valores, nombres de clases, etc. la mayoría de las cosas que sean formadas por palabras las podremos manipular usando strings.

Los números nos servirán para realizar tanto lógica como para manipular valores o atributos de los elementos del DOM. Vamos a poder construir lógica para realizar acciones dependiendo del resultado de los cálculos de estos números, por ejemplo, contar la cantidad de letras en un input, o realizar cambios en un elemento cuando el ancho de la página sea menor que cierto número.

Los Booleanos serán usados también en lógica, verificando, por ejemplo, si una operación es verdadera o no, si un elemento existe o no en el DOM, una infinidad de operaciones nos pueden dar un resultado final VERDADERO o FALSO, SÍ o NO, EXISTE o NO EXISTE.

Es por esto, lo importante que es conocer los tipos de datos en JavaScript, para así saber que tipo de datos estamos manipulando al programar.

Variables y Constantes

Competencias

- Entender qué es una variable y una constante.
- Aprender a utilizar variables y constantes para almacenar nuestros valores.
- Aplicar variables y constantes en ejemplos prácticos.

Introducción

Como ya lo hemos mencionado en más de una ocasión dentro de la unidad, programación es la manipulación de datos. Hemos visto que existen distintos tipos de datos los cuales podemos manipular y hacer muchas cosas con ellos.

Ahora bien, en ciertas ocasiones para realizar distintas operaciones necesitaremos almacenar de alguna manera algunos de estos datos que estamos manipulando.

Variables

Lo que la programación nos brinda para que podamos almacenar datos, son las **variables.**

Para que entendamos de una manera más práctica, las variables son como cajas. Si, como cajas donde podemos guardar datos.

Por ejemplo, si tenemos que guardar el nombre del usuario que ingreso a la plataforma de desafío latam, para luego imprimirlo por pantalla y así el usuario pueda ver su nombre, lo podemos guardar en una variables en alguna “cajita”.

Lo importante es entender que una variable es una caja donde podemos meter datos.

Las variables en JavaScript se declaran de la siguiente manera:

```
var usuario = 'Alexis';
var edad = 30;
```

Se declaran con la palabra “var”, este “var” se encarga de decirle a JavaScript o al computador que lo que viene es una variable. Luego seguido de “var” viene el nombre que queremos colocarle a la variable, a continuación después del signo **=** viene el tipo de dato que estamos almacenando.

Para poder utilizar una variable hay miles de formas, todo va a depender de la manera que nosotros creemos que es más conveniente manipularla.

Como su nombre lo dice, una variable su valor va o puede variar. Vamos a hacer un ejemplo sobre esto para que podamos entender.

- Como aprendimos en el capítulo anterior, ahora crearemos un nuevo archivo html y le incluiremos el archivo .js donde agregaremos el código Javascript.
- Al archivo .html lo llamaremos como “variables.html” y a nuestro .js le pondremos por nombre “variables.js”
- Una vez incluido nuestro archivo “variables.js” en el html, escribiremos el siguiente código:

```
var nombre = "Alexis";
alert(nombre);
```

Lo ejecutamos y obtendremos un mensaje de alerta con el valor que le asignamos a nuestra variable. En este caso el valor fue de tipo de texto (string) que decía “Alexis”.

- Ahora borramos y declaramos la misma variable pero no le asignamos ningún valor (creamos nuestra caja la dejamos abierta pero no colocamos nada dentro de ella por el momento)

```
var nombre;
```

- Agregamos también otra variable, la cual almacenará una edad:

```
var edad;
```

No le asignamos a ninguna de las dos variables un valor, ya que en el ejemplo queremos ver el por qué del nombre “variable”. El valor se lo asignaremos nosotros cuando carguemos la página .html en el navegador.

Para hacer esto, agregamos la siguiente línea de código:

```
nombre = prompt("Ingrese su nombre");
edad = prompt("Ingrese su edad");

// Le asignamos el valor a cada variable desde el valor que ingresamos en el
método prompt.
```

Lo que el método “prompt();” nos permite hacer aquí, es solicitar que el usuario ingrese un dato, luego ese dato ingresado lo almacenamos en cada una de las variables que declaramos.

Luego, utilizando el comando document.write(); imprimimos en la página .html los valores de cada variable de la siguiente manera:

```
var nombre;
var edad;

nombre = prompt("Ingrese su nombre");
edad = prompt("Ingrese su edad");

document.write(nombre + " " + edad);
// Concatenamos las dos variables y las separamos con un espacio en blanco
```

En el comando document.write(); lo que hicimos fue mostrar los valores ingresados y almacenados en cada variable. El operador + nos sirve para concatenar o unir dos valores distintos.

En este ejemplo pudimos apreciar básicamente la función de las variables, esa función de almacenar datos para su manipulación. Pudimos observar que este valor es variable, cambia cada vez que en este caso nosotros cargamos la página .html e ingresamos datos.

En las variables también podemos realizar operaciones matemáticas entre ellas, por ejemplo:

```
var num1 = 10;
var num2 = 20;

var resultado = num1 + num2;

document.write("La suma de los números" + " " + num1 + " " + " y " + num2 + "
" + "es:" + " " + resultado);
```

En las variables también podemos almacenar operaciones matemáticas, por ejemplo:

```
var multiplicacion = 5 * 9;

document.write("Resultado de multiplicación: " + multiplicacion);
```

Las variables son de muchas ayuda y una gran herramienta al momento de programar y manipular datos, por el contrario, cuando no se requiere que estas variables se modifiquen, se utiliza el recurso de las “Constantes”.

Constantes

Las constantes son parecidas a las variables ya que almacenan datos, con la diferencia que no pueden cambiar porque son “constantes”. Una variable puede tener un valor, pero luego puede tener otro, y así sucesivamente, pero, cuando estemos programando vamos querer en muchos casos almacenar datos los cuales no queremos que cambien, y para esto utilizamos las constantes.

Cuando yo declaro una constante esta ya no es modificable, sino que solamente tiene carácter de lectura.

La forma en que nosotros podemos declarar una constante es de la siguiente manera:

```
const añoNacimiento = 1988;
```

En vez de “var” como en las variables, acá utilizamos la palabra reservada “const” seguido del nombre que le pondremos a la constante y posterior su valor.

En el siguiente ejemplo básico veremos cómo utilizar una constante. Y la diferencia entre una variable. Haremos una operación matemática con un valor constante que nunca cambia y con un dato variable.

Creamos al igual que las veces anteriores nuestro archivo .html y nuestro archivo JavaScript. Agregamos el siguiente código a nuestro “.js” :

```
const pi = 3.14;
var num;

num = prompt("Ingrese número para multiplicar por 'pi': ");
alert(pi*num);
```

¿Qué hicimos? Simple, declaramos una constante llamada “pi” la cual matemáticamente tiene un valor el cual nunca cambia, siempre “pi” va a tener un valor fijo.

También declaramos una variable “num”, la cual obtendrá su valor de acuerdo a lo que nosotros ingresemos en pantalla con el método “prompt” que usamos anteriormente.

Comentario: “En la versión anterior a ES6 existía solamente una manera de declarar variables, que era con la palabra “var”, como lo vimos anteriormente. Ahora con ES6 (ECMAScript 6) tenemos dos opciones más: “const” y una manera que no vimos pero que brevemente explicaremos aquí, que es “let”.

“var”, en resumen, tiene ámbito de función. Funciona y la podemos llamar dentro y hacia una función. Pero con “let” podemos crear variables en ámbitos de bloques. Es decir, por ejemplo, declaramos una variable let i en un ciclo for, y esta variable solamente podremos ocupar dentro del ciclo for. Por ejemplo:

```
if (true){
    let a = 'hola';
    console.log(a);
}
```

Aquí, imprimimos por consola el valor de la variable a, declarada con “let”. Dentro del bloque del if funciona perfectamente. Pero si imprimimos a la misma variable desde fuera del bloque if, no nos funcionara:

```
if (true){
    let a = 'hola';
}
    console.log(a);
```

Y “const” también tiene ámbito de bloques, pero a diferencia de “var” y “let” no puede ser re-asignado su valor.

Debido a que este es un primer acercamiento al mundo de la programación, las ventajas y desventajas entre var, let y const se abordarán en las siguientes unidades a lo largo de la carrera.