Desarrollo
Front-End

Universidad Nacional del Nordeste
Facultad de Ciencias Exactas, Naturales y Agrimensura

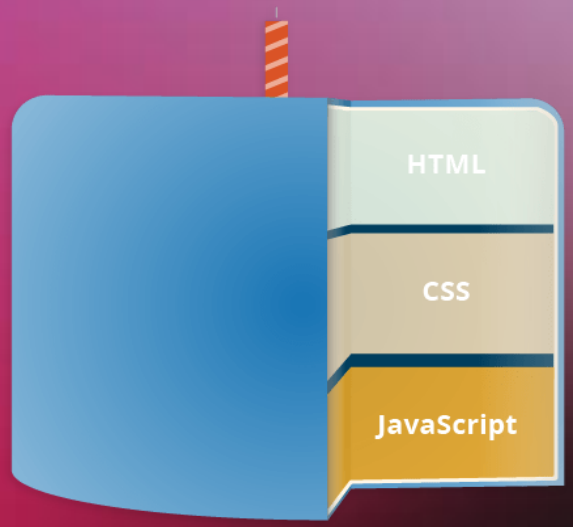
Argentina programa 4.0

Módulo 4. Introducción a Javascript

- Introducción
  - Que es Javascript?, ¿qué puede hacer realmente?.
  - Variables, Constantes, Operadores, Matemática básica, Manejar texto, Array,...
- Estructuras dentro de Javascript
  - If, Else, Else-if, For
- Funciones
  - Integradas del navegador, Mi propia función
  - Ámbito de la función y conflictos

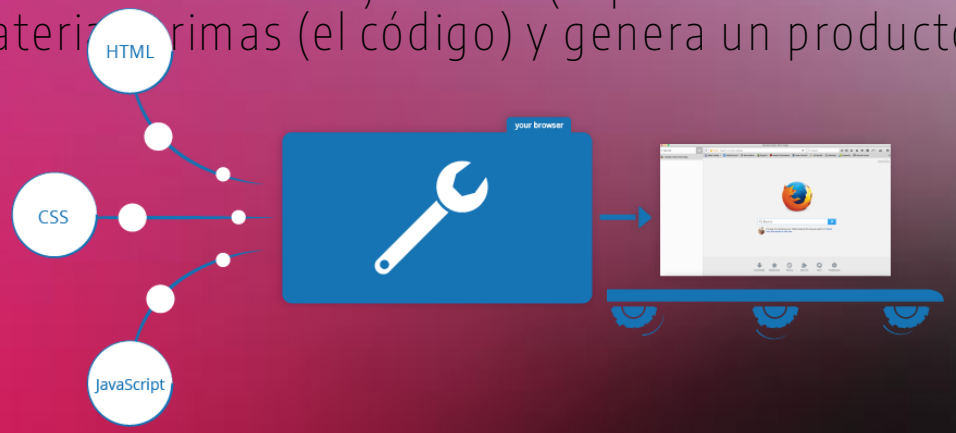
¿Qué es Javascript?

- Es un lenguaje de programación o de secuencias de comandos que te permite implementar funciones en páginas web.
- Muestra oportunas actualizaciones de contenido, mapas interactivos, animación de Gráficos 2D/3D, desplazamiento de máquinas reproductoras de vídeo, etc.,
- Es la tercera capa de las tecnologías web estándar, dos de las cuales (HTML y CSS) hemos explicado en anteriores clases.



Qué puede hacer realmente

- El núcleo del lenguaje JavaScript de lado del cliente consta de algunas características de programación comunes que te permiten hacer cosas como:
  - Almacenamiento de valores utiles dentro de variables.
  - Operaciones sobre fragmentos de texto (conocidas como "cadenas" (strings) en programación).
  - Entre otras cosas mas.
- ¿Qué está haciendo JavaScript en tu página?
  - Cuando cargas una página web en tu navegador, estás ejecutando tu código (HTML, CSS y JavaScript) dentro de un entorno de ejecución (la pestaña del navegador). Esto es como una fábrica que toma materias primas (el código) y genera un producto (la página web).



Variables y Constantes

- ¿Qué es una Variable?
  - Una variable es un contenedor para un valor, es un espacio en memoria donde se almacena valores.
  - Los valores que contienen pueden cambiar.
  - Para usar una variable, primero debes crearla — precisamente, a esto lo llamamos declarar la variable. Para hacerlo, escribimos la palabra clave var o let seguida del nombre con el que deseas llamar a tu variable:
    - *Ejemplo: Let nombre = "Martin" o Var nombre = "Martin"*
  - Diferencia entre var y let
    - La diferencia principal entre **var** y **let** en JavaScript está en el alcance y la forma en que se manejan las variables.
    - **let** es una forma más moderna y recomendada de declarar variables en JavaScript, ya que proporciona un alcance de bloque más estricto y evita algunos errores comunes. Se recomienda utilizar let en lugar de var en la mayoría de los casos, a menos que haya una necesidad específica de utilizar el alcance de función o se esté trabajando con código más antiguo que ya utiliza var.
- **¿Qué es una Constante?**
  - Una constante es un identificador (nombre) que se utiliza para representar un valor que no cambia durante la ejecución de un programa. Una vez que se asigna un valor a una constante, ese valor no puede modificarse posteriormente.
  - La declaración de una constante se realiza utilizando la palabra clave const.
    - Ejemplo: const PI = 3.14159;

Operadores

- Un operador es básicamente un símbolo matemático que puede actuar sobre dos valores (o variables) y producir un resultado.

Operador	Explicación	Símbolo(s)	Ejemplo
Suma/concatena	Se usa para sumar dos números, o juntar dos cadenas en una.	+	6 + 9; "Hola " + "mundo!";

Resta, multiplicación, división	Estos hacen lo que esperarías que hicieran en las matemáticas básicas.	-, *, /	9 - 3;
			8 * 2; // La multiplicación en JS es un asterisco
			9 / 3;

Operador de asignación	Los has visto anteriormente: asigna un valor a una variable.	=	let miVariable = 'Bob';
identidad/igualdad	Comprueba si dos valores son iguales entre sí, y devuelve un valor de true/false (booleano).	===	let miVariable = 3; miVariable === 4;

Negación, distinto (no igual)	En ocasiones utilizado con el operador de identidad, la negación es en JS el equivalente al operador lógico NOT — cambia true por false y viceversa.	!, !==	La expresión básica es true, pero la comparación devuelve false porque lo hemos negado:
			let miVariable = 3;
			!miVariable === 3;
			Aquí estamos comprobando "miVariable NO es igual a 3". Esto devuelve false, porque miVariable ES igual a 3.
			let miVariable = 3;
			miVariable !== 3;

Matemática básica

- Tipos de números:
  - Enteros son números sin parte decimal, e.g. 10, 400 o -5.
  - Números con coma flotante (floats): tienen punto decimal y parte decimal, por ejemplo,12.5 y 56.7786543.
  - Doubles: son un tipo específico de números de coma flotante que tienen una mayor precisión que los numeros de coma flotante comunes (lo que significa que son más precisos en cuanto a la cantidad de decimales que poseen).

  ¡Incluso tenemos distintos tipos de sistemas numéricos! El decimal es base 10 (quiere decir que utiliza 0-9 en cada columna), pero también tenemos cosas como:

  - Binario — El lenguaje de computadora de nivel más bajo; 0s y 1s.
  - Octal — De base 8, utiliza de 0–7 en cada columna.
  - Hexadecimal — De base 16, utiliza de 0–9 y luego de a–f en cada columna. Puedes haber encontrado estos números antes, cuando colocabas colores en CSS.

Matemática básica

- Operadores Aritméticos

Operador	Nombre	Propósito	Ejemplo
+	Adición	Suma dos números juntos.	6 + 9
-	Resta	Resta el numero de la derecha del de la izquierda.	20 - 15
*	Multiplicación	Multiplica dos números juntos.	3 * 7
/	División	Divide el número de la izquierda por el de la derecha.	10 / 5
%	Sobrante (también llamado módulo)	Retorna el restante después de dividir el número de la izquierda en porciones enteras del de la derecha.	8 % 3 (retorna 2, como tres está dos veces en 8, quedando 2 restantes.)

Operadores Lógicos

Operador	Nombre	Propósito	Ejemplo
&&	Operador **Y**	Se emplea cuando en una estructura condicional se disponen dos condiciones	(num1 > num2) && (num1 > num3)
||	Operador **O**	Cuando vinculamos dos o más condiciones con el operador "O", con que una de las dos condiciones sea Verdadera alcanza para que el resultado de la condición compuesta sea Verdadero.	(mes == 1) || (mes == 2)

Estructuras dentro de Javascript

Estructura condicional simple (IF)

Es una expresión que se evalúa como verdadera o falsa. Si la condición es verdadera, el bloque de código dentro de las llaves "{}" se ejecutará. Si la condición es falsa, ese bloque se omitirá y el programa continuará ejecutando las instrucciones después del bloque "if".

```
if (condicion) {
  // Código a ejecutar si la condición es verdadera
}
```

EJEMP

```
let edad = 20;

if (edad >= 18) {
  console.log("Eres mayor de edad.");
}
```

Estructura condicional compuestas (IF – Elseif – Else)

Las estructuras condicionales compuestas se utilizan para ejecutar diferentes bloques de código dependiendo de múltiples condiciones. La estructura más común para esto es la declaración "if-else if-else". La sintaxis general es la siguiente:

```
if (condicion1) {
    // Código a ejecutar si condicion1 es verdadera
} else if (condicion2) {
    // Código a ejecutar si condicion1 es falsa y condicion2 es verdadera
} else {
    // Código a ejecutar si ninguna de las condiciones anteriores es verdader
}
```

EJEMPLO

```
var edad = 18;

if (edad < 18) {
    console.log("Eres menor de edad.");
} else if (edad >= 18 && edad <= 65) {
    console.log("Eres adulto.");
} else {
    console.log("Eres mayor de 65 años.");
}
```

Estructuras dentro de Javascript

Estructura repetitiva (for)

Es un bucle que permite ejecutar un bloque de código varias veces. Se utiliza principalmente cuando se conoce de antemano el número exacto de iteraciones que se deben realizar. La sintaxis general del bucle "for" en JavaScript es la siguiente:

```
for (inicialización; condición; incremento/decremento) {
  // Código a ejecutar en cada iteración
}
```

La "inicialización" se utiliza para establecer un valor inicial antes de comenzar el bucle. La "condición" es una expresión que se evalúa antes de cada iteración, y mientras sea verdadera, se ejecutará el bloque de código. El "incremento/decremento" se utiliza para modificar el valor de control en cada iteración.

EJEMPLO

```
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

En este ejemplo, la variable "i" se inicializa en 1. La condición establece que el bucle debe continuar mientras "i" sea menor o igual a 5. Después de cada iteración, se incrementa el valor de "i" en 1. En cada iteración, se imprime el valor de "i" utilizando la función "console.log()". Esto resultaría en la impresión de los números del 1 al 5 en la consola.

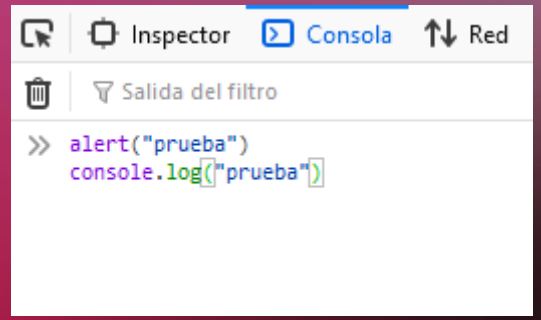
Funciones

Integradas del navegador

Son funciones predefinidas que están disponibles en el entorno del navegador web y se pueden utilizar para realizar diferentes tareas. Estas funciones son proporcionadas por el propio navegador y no requieren ninguna importación o configuración adicional. Algunas de las funciones integradas más comunes en JavaScript que están disponibles en el navegador son:

1.alert(): Muestra un cuadro de diálogo con un mensaje y un botón de "Aceptar".
2.confirm(): Muestra un cuadro de diálogo con un mensaje y dos botones, "Aceptar" y "Cancelar". Devuelve true si el usuario hace clic en "Aceptar" y false si hace clic en "Cancelar".
3.prompt(): Muestra un cuadro de diálogo con un mensaje, un campo de entrada y dos botones, "Aceptar" y "Cancelar". Devuelve el valor ingresado por el usuario si hace clic en "Aceptar" y null si hace clic en "Cancelar".
4.console.log(): Escribe un mensaje en la consola del navegador para fines de depuración y registro de información.
5.setTimeout(): Ejecuta una función o un bloque de código después de un cierto tiempo especificado.
6.setInterval(): Ejecuta una función o un bloque de código repetidamente cada cierto tiempo especificado.

Podemos probar estas funciones apretando la tecla F12 del teclado, dirigiéndonos a la pestaña “Consola”



Funciones - Mi propia función

Una función es un conjunto de instrucciones que resuelven una parte del problema y que puede ser utilizado (llamado) desde diferentes partes de un programa.

- Consta de un nombre y parámetros. Con el nombre llamamos a la función, es decir, hacemos referencia a la misma.
- Los parámetros son valores que se envían y son indispensables para la resolución del mismo.
- La función realizará alguna operación con los parámetros que le enviamos. Podemos cargar una variable, consultarla, modificarla, imprimirla, etc.

Ejemplo: hemos definido una función llamada "saludar" que toma un parámetro llamado "nombre". Dentro de la función, imprimimos un saludo utilizando el valor del parámetro "nombre" concatenado con el texto adicional.

```
function saludar(nombre) {
  console.log("¡Hola, " + nombre + "!");
}

// Llamada a la función
saludar("Juan");
```

Para utilizar la función, simplemente la llamamos pasando un argumento.

Funcion - Ámbito de la función y conflictos

El ámbito de una función determina la visibilidad y accesibilidad de las variables y funciones dentro de esa función. Un conflicto puede ocurrir cuando se intenta acceder a una variable o función que no está definida en el ámbito actual o cuando hay nombres de variables o funciones que se solapan.

Existen dos tipos principales de ámbito

Ámbito global: Las variables y funciones declaradas fuera de cualquier función tienen un ámbito global. Estas variables y funciones son accesibles desde cualquier parte del código, tanto dentro de funciones como fuera de ellas.

```
var x = 10;

function imprimirX() {
  console.log(x);
}

imprimirX(); // Imprime 10
```

Ámbito local: Las variables y funciones declaradas dentro de una función tienen un ámbito local. Estas variables y funciones solo son accesibles dentro de la función en la que se declaran.

```
function sumar(a, b) {
  var resultado = a + b;
  console.log(resultado);
}

sumar(2, 3); // Imprime 5
console.log(resultado); // Genera un error: resultado is not defined
```

La variable "resultado" está declarada dentro de la función "sumar" y solo es accesible dentro de esa función. Si intentas acceder a ella fuera de la función, se generará un error.

Funcion - Ámbito de la función y conflictos

En cuanto a los conflictos, pueden ocurrir cuando se utilizan nombres de variables o funciones que se solapan en diferentes ámbitos. Esto puede generar resultados inesperados o errores en el código.

```
var x = 10;

function imprimirX() {
  var x = 5;
  console.log(x);
}

imprimirX(); // Imprime 5
console.log(x); // Imprime 10
```

Hay una variable "x" declarada en el ámbito global y otra variable "x" declarada dentro de la función "imprimirX". Al imprimir el valor de "x" dentro de la función, se obtiene el valor local (5), pero al imprimirlo fuera de la función, se obtiene el valor global (10).
Para evitar conflictos de nombres, es recomendable utilizar nombres de variables y funciones que sean descriptivos y únicos en cada ámbito. También es posible utilizar técnicas como el uso de módulos o el encapsulamiento para evitar la contaminación del ámbito global y reducir la posibilidad de conflictos.