"MUJERES DIGITALES 2024"

AHK Deutsch-Kolumbianische Industrie- und Handelskammer
Cámara de Industria y Comercio Colombo-Alemana

Key code

SENA

"MUJERES DIGITALES 2024"

AHK Deutsch-Kolumbianische Industrie- und Handelskammer
Cámara de Industria y Comercio Colombo-Alemana

Key code

Repaso JavaScript
Clase 2

Arreglos

Introducción:

1. ¿Qué es un Arreglo?:

- **Definición:** Un arreglo es una colección de elementos que se almacenan en una sola variable.

Arreglos

2. Creación de un Arreglo:

- Imagina que quieres crear una lista de tus frutas favoritas. Un arreglo te permite hacer eso de una manera muy simple.

```
let frutas = ["Manzana", "Banana", "Naranja"];
```

Arreglos

3. Acceso a Elementos en un Arreglo

- Pueden acceder a cada elemento de un arreglo usando su posición (índice). Los índices en un arreglo empiezan en 0, lo que significa que el primer elemento se cuenta desde el 0.

```
console.log(frutas[0]);
```

```
console.log(frutas[1]);
```

Arreglos

4. Modificación de Elementos en un Arreglo

- También pueden cambiar un elemento específico en un arreglo.

```
frutas[1] = "Pera";
console.log(frutas);
```

Arreglos

5. Métodos Comunes de Arreglos

Aquí veremos algunos métodos que nos permiten agregar, eliminar y contar elementos en un arreglo.

- push() - Agrega un nuevo elemento al final del arreglo.

```
frutas.push("Mango");
console.log(frutas);
```

Arreglos

- **pop()** - Elimina el último elemento del arreglo.

```
frutas.pop();
console.log(frutas);
```

- **shift()** - Elimina el primer elemento del arreglo.

```
frutas.shift();
console.log(frutas);
```

Arreglos

- **unshift()** - Agrega un nuevo elemento al principio del arreglo.

```
frutas.unshift("Fresa");
console.log(frutas); //
```

- **length** - Devuelve el número de elementos en un arreglo.

```
console.log(frutas.length);
```

Arreglos

6. Ejercicios:

Ejercicio 1:

- Crea un arreglo llamado `colores` con al menos 4 colores.
- Imprime en la consola el primer y el último color del arreglo.

Ejercicio 2:

- Agregar otro color al arreglo.
- Elimina el primer color
- Muestra cuántos colores hay en el arreglo

Funciones

Introducción:

1. ¿Qué es una Función?:

- **Definición:** Imagina que una función es como una receta de cocina. La receta (función) te dice exactamente qué hacer paso a paso.

Funciones

2. ¿Por qué Usar Funciones?

- Las funciones nos permiten reutilizar código.

3. Cómo Crear una Función

1. Funciones Declarativas:

- Esta es la forma más común de crear una función. Es como darle un nombre a la receta y luego escribir los pasos.

Funciones

Ejemplo:

```
// Declarar una función para saludar
function saludar(nombre) {
    return "Hola, " + nombre + "!";
}

// Usar (llamar) la función
console.log(saludar("María")); // Output: "Hola, María!"
```

Funciones

2. Funciones de Expresión

- Otra forma de crear una función es asignarla a una variable. Es como guardar la receta en un cajón y usarla cuando la necesites.

```
// Crear una función que suma dos números
const sumar = function(a, b) {
    return a + b;
};

// Usar la función
console.log(sumar(3, 4)); // Output: 7
```

Funciones

4. Parámetros y Retorno

- **Parámetros:**

Los parámetros son como los ingredientes de la receta. Son los valores que la función necesita para hacer su trabajo.

```
// Función que multiplica dos números
function multiplicar(a, b) {
    return a * b;
}

console.log(multiplicar(2, 5)); // Output: 10
```

Funciones

Retorno:

- El retorno es como el plato terminado de la receta. Es el resultado que obtenemos después de que la función hace su trabajo.

```
// Función que resta dos números
function restar(a, b) {
    return a - b;
}

console.log(restar(10, 4)); // Output: 6
```

Funciones

5. Funciones Anónimas y Arrow Functions:

1. Funciones Anónimas

Son funciones sin nombre que se guardan en una variable.

```
// Crear una función anónima para sumar dos números
const suma = function(a, b) {
    return a + b;
};

console.log(suma(2, 3)); // Output: 5
```

Funciones

2. Arrow Functions (Funciones Flecha)

Son una forma más rápida y moderna de escribir funciones.

```
// Crear una arrow function para sumar dos números
const sumarFlecha = (a, b) => {
    return a + b;
};

console.log(sumarFlecha(4, 6)); // Output: 10
```

Funciones

Ejercicios:

1. **Crear una función de saludo:**

- Crea una función llamada `darBienvenida` que tome un nombre y devuelva un mensaje de bienvenida.

- Ejemplo: `darBienvenida("Ana")` debería devolver `"¡Bienvenida, Ana!"`.

2. Crear una función de suma:

- Crea una función llamada `sumarTresNumeros` que tome tres números y devuelva su suma.

- Ejemplo: `sumarTresNumeros(2, 3, 4)` debería devolver `9`.

Introducción a Git

Introducción:

1. ¿Qué es Git?:

Definición: Imagina que estás escribiendo un libro. Cada vez que haces un cambio en el texto, te gustaría poder guardar una versión para poder volver a ella si algo sale mal

Introducción a Git

2. Conceptos Básicos de Git

1. Repositorio

Un repositorio es como una carpeta donde Git guarda todas las versiones de tu proyecto. Puede estar en tu computadora (local) o en internet (remoto, como en GitHub).

2. Comandos Básicos de Git

Vamos a ver algunos comandos esenciales que te ayudarán a empezar a usar Git.

Introducción a Git

3. Comandos Básicos

1. `git init`

- Este comando crea un nuevo repositorio en tu computadora.

- **Ejemplo:**

```
git init
```

Introducción a Git

2. `git add`

- Este comando le dice a Git que quieres incluir los cambios de un archivo específico en la próxima "foto" (commit) de tu proyecto.

- **Ejemplo:**

```
git add archivo.txt
```

Introducción a Git

3. `git commit`

- Este comando crea una "foto" de tu proyecto en su estado actual.

- **Ejemplo:**

```
git commit -m "Mensaje del commit"
```

Introducción a Git

4. `git status`

- Este comando te muestra el estado actual de los archivos en tu repositorio.

- **Ejemplo:**

```
git status
```

Introducción a Git

4. Creación de un Repositorio Local y Primer Commit

- Imagínate que estás trabajando en un proyecto y quieres empezar a usar Git para controlar las versiones.

Paso 1: Crear una carpeta para el proyecto Primero, crea una carpeta para tu proyecto en tu computadora.

```
mkdir mi_proyecto
cd mi_proyecto
```

Introducción a Git

Paso 2: Iniciar Git en la carpeta Ahora que estás dentro de la carpeta, inicia Git con el siguiente comando:

```
git init
```

- Esto crea un repositorio de Git vacío en tu carpeta.

Paso 3: Crear un archivo y añadirlo a Git Crea un archivo dentro de esta carpeta, por ejemplo, `index.html`.

```
echo "<h1>Hola, mundo</h1>" > index.html
```

Introducción a Git

- Ahora, añádelo al control de versiones con Git:

```
git add index.html
```

Paso 4: Hacer el primer commit Guarda la versión actual de tu proyecto con un commit.

```
git commit -m "Primer commit: Añadir index.html"
```

- Ahora has guardado tu primer "foto" de tu proyecto.

Introducción a Git

5. Ejercicio Práctico:

- **Crea un repositorio local para un proyecto sencillo:**
  - Sigue los pasos anteriores para crear una carpeta, iniciar Git, y hacer un commit.
- **Experimenta con los comandos básicos:**
  - Haz cambios en tu archivo `index.html`, como cambiar el texto, y usa `git add`, `git commit`, y `git status` para practicar cómo guardar esos cambios.

Buenas Prácticas en Git

1. Introducción

- **¿Qué es un Commit?:**

Un commit es como tomar una "foto" de tu proyecto en un momento específico.

Buenas Prácticas en Git

2. Estructuración de Commits

Imagina que estás escribiendo un libro:

- Cada commit es como guardar una versión de tu libro después de escribir un capítulo.

Buenas Prácticas en Git

1. Haz Commits Pequeños y Frecuentes

- **Por qué:** Es más fácil rastrear cambios y encontrar errores cuando los commits son pequeños y específicos.

- **Ejemplo:** Si estás trabajando en una página web, podrías hacer un commit después de terminar la estructura HTML, otro después de añadir estilos CSS, y otro después de escribir el código JavaScript.

```
git commit -m "Añadir estructura básica HTML"
git commit -m "Añadir estilos CSS para la cabecera"
git commit -m "Implementar interacción con JavaScript en el botón"
```

Buenas Prácticas en Git

2. Agrupa Cambios Relacionados

- **Por qué:** Mantén juntos los cambios que están relacionados. No combines en un solo commit cambios en la estructura HTML con la adición de nuevas imágenes, por ejemplo.

- **Ejemplo:** Si estás arreglando un error y añadiendo una nueva función, haz dos commits separados:

```
git commit -m "Corregir error de alineación en el footer"
git commit -m "Añadir funcionalidad de búsqueda en el header"
```

Buenas Prácticas en Git

3. Uso de Mensajes de Commit Claros y Concisos

- **Imagina que estás dejando notas para ti mismo o para alguien más que trabajará en el mismo proyecto.** Los mensajes de commit deben explicar claramente qué cambios has hecho y por qué.

Buenas Prácticas en Git

1. Escribe Mensajes Claros

- **Por qué:** Ayuda a los demás (y a ti mismo en el futuro) a entender rápidamente qué cambios se hicieron.

- **Ejemplo:** En lugar de escribir "cambios en index.html", escribe algo como "Añadir sección de testimonios en index.html".

Buenas Prácticas en Git

2. Usa el Modo Imperativo

- **Por qué:** Es una convención común en Git escribir los mensajes de commit como si fueran órdenes. Esto hace que los mensajes sean consistentes y fáciles de leer.
- **Ejemplo:** "Añadir", "Corregir", "Eliminar", "Actualizar" son ejemplos de verbos en imperativo.

```
git commit -m "Añadir validación de formulario en contacto.html"
git commit -m "Corregir enlace roto en el menú de navegación"
```

Buenas Prácticas en Git

3. Explica el Porqué Cuando Sea Necesario

- **Por qué:** A veces es importante explicar no solo qué hiciste, sino por qué lo hiciste.

- **Ejemplo:** Si cambias algo que no es obvio, añade una breve explicación.

```
git commit -m "Actualizar librería de JavaScript a la última versión para corregir problemas de compatibilidad"
```

Buenas Prácticas en Git

Ejercicio

Paso 1: Crear un Archivo de Texto Sencillo

1. Crea un archivo de texto simple llamado `notas.txt` en una carpeta de tu computadora.
   - Puedes usar un editor de texto simple como Notepad, TextEdit, o cualquier editor de código.
2. Escribe una frase en el archivo como, por ejemplo: "Este es mi primer archivo de notas".

Buenas Prácticas en Git

Paso 2: Iniciar un Repositorio con Git

- **Abre la terminal** (o la línea de comandos) y navega a la carpeta donde guardaste el archivo `notas.txt`.
- **Inicia un repositorio de Git** en esa carpeta usando el siguiente comando:

```
git init
```

Buenas Prácticas en Git

- **Añade el archivo a Git** para rastrear los cambios:

```
git add notas.txt
```

- **Haz tu primer commit:**

```
git commit -m "Añadir archivo de notas con la primera frase"
```

Buenas Prácticas en Git

Paso 3: Hacer Pequeños Cambios y Commits

- **Abre el archivo `notas.txt`** nuevamente y añade otra frase como: "Esta es la segunda frase".
- **Guarda el archivo** y luego añade y comitea los cambios:

```
git add notas.txt

git commit -m "Añadir segunda frase a las notas"
```

- **Repite el proceso** una o dos veces más:
  - Escribe nuevas frases o modifica las existentes.
  - Guarda los cambios, añade el archivo a Git y haz un commit.

Buenas Prácticas en Git

Paso 4: Revisar el Historial de Commits

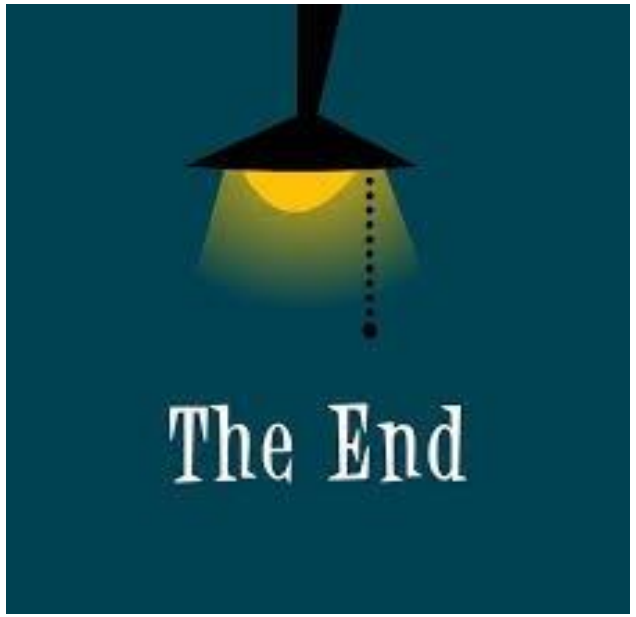
- **Usa el comando `git log`** para ver el historial de los commits que has hecho:

```
git log
```

- **Revisa los mensajes de commit**:
  - Evalúa si los mensajes que escribiste son claros y describen adecuadamente los cambios que realizaste en el archivo `notas.txt`.

"MUJERES DIGITALES 2024"

¡¡¡Muchas Gracias!!!



Buenas Prácticas en Git