Clase 0 Introducción a la programación (Ver Buenas prácticas)

Escribiendo buen código

Normas para lograr obtener un buen código.

Nos preguntaríamos en programación ¿Porqué?¿No basta con que funcione?

La respuesta es **no** del todo, debemos tener en cuenta que un buen código:

- *Facilita* encontrar los inevitables errores
- Hace más *cómodo* modificar el script
- Hace más fácil *actualizar* los scripts
- *Otros* programadores también pueden trabajar en el programa

Los programas en javascript deben se**r fácilmente entendibles**.

¿Y esto como se consigue? Pues siguiendo unas **buenas prácticas en programación**, que no solo valen para Javascript, sino para cualquier otro lenguaje.

La programación tiene un punto de arte y pueden existir múltiples soluciones para un mismo problema. Esto hace que cada programador tenga un cierto estilo que lo caracteriza. Pero en general hay algunas reglas o tips que ayudan a **conseguir un buen código**.

Primero proyecta

Antes de ponerte a escribir código Javascript, escribe tu programa usando lenguaje cotidiano, lo que se conoce como algoritmo o pseudocódigo. Te permitirá tener una idea clara de lo que vas a hacer y te facilita la buena documentación de tus programas.

Divide para vencer

No quieras resolver el todo de una. Intenta dividir en pequeñas tareas. Te será más fácil y encontraras una solución más eficiente que si tratas de solucionarlo todo de una.

Comentarios de código

Todos los lenguajes de programación poseen algún sistema para insertar comentarios en los programas. Sirven para explicar lo que hace el programa. No debe faltar por ejemplo al crear una función o un objeto. Estos solo lo ve el programador.

Comentarios de datos

No solo el código debe ser explicado, también se debe dejar claro para que se van a usar las variables, con comentarios explícitos o con nombres significativos.

Código autodocumentado

Hay que hacer el código claro. Llamar a las funciones y variables con nombres relativos a lo que hacen. Si defines una variable para guardar una altura no la llames x, llámala altura. si usas una función para abrir una ventana llámala abreVentana() por ejemplo.

Líneas e instrucciones

Para tener un código claro y fácil de analizar usa una linea para cada instrucción. No crees bloques de lineas muy compactos que haga difícil un análisis visual.

Indentados

Usa las indentaciones o tabulaciones, espacios en blanco delante de las lineas del programa, para destacar bloques de código como definiciones de funciones o bloques condicionales o cuerpos de bucles.

Variables

Trabaja en modo estricto: usa el punto y coma al final de cada instrucción y declara las variables que vayas a usar.

Variables globales

Minimiza el uso de variables globales (esas que son accesibles desde cualquier parte del programa).

Prever

Si hay que ser muy torpe para provocar un determinado error, ten por seguro que habrá alguien con ese grado de torpeza. Así que siempre que veas un punto donde pueda producirse un error, prevé que hacer cundo se produzca.

Refactoriza

Al terminar tu programa vuelve a revisarlo para eliminar posibles errores o mejorar algunas soluciones o completar los comentarios.... Es como darle el pulido final.

Todos estos puntos son opcionales, puedes seguirlo o no. Pero un buen código no solo denota un buen programador sino que logra programas más eficientes.

Javascript no es nada estricto en sus normas.

Considera el *cambio de linea* como el fin de una instrucción y las variables pueden ser usadas sin una declaración (definición) previa. Detalles como estos hacen que muchos vean Javascript como un lenguaje poco serio.

No obstante existe un *modo estricto* ("*use strict*") que te va a obligar, por ejemplo a usar los *puntos y coma* y a *declarar* las variables antes de usarlas.

Será tu decisión el tener en cuenta estos ítem o no a la hora de programar, recuerda que algún momento podrías enfrentarte a realizar programas con muchas líneas de código, en ese momento te será útil estos puntos.

Ejemplos de lo anterior

Esta función determina si un número es divisible por otro:

function x(a,b){ return a%b==0}

Muy bueno, en una línea de código realizamos todo, estupendo, ***pues no tanto****.*

```
function esDivisor(numero, divisor){
    var esdivisor; //No podemos dividir por 0
    if (divisor==0){
        esdivisor = "error, el divisor es 0";
    }
    else
    //Si el resto es cero, entonces es un divisor
        esdivisor = (numero % divisor == 0);
        esdivisor = "El número es divisor";
        return esdivisor;
    }
```

Actividad
¿Qué ítem se usaron para mejorar este código?

Existen reglas básicas que deben cumplirse y otras no obligatorias pero si recomendables.

El programa consta de una serie de órdenes o **sentencias** y **comentarios.**

Los comentarios que ocupan una sola linea se indican mediante dos barras inclinadas **//**. si se necesitan más lineas se usa una marca de inicio **/*** y una de cierre ***/**.

Las sentencias son órdenes para que el programa **ejecute** alguna acción o para **controlar** el orden en que se ejecutan las acciones del programa.

Existen sentencias de

- **asignación**, cuando se guarda un valor en una **variable.** El valor puede ser el resultado devuelto por una función o un valor literal.
- **Invocación** de funciones o expresiones. Las expresiones devuelven valores y las funciones pueden devolver valores o no.
- **Control** controlan la ejecución o no de otras sentencias

Las sentencias pueden agruparse en **bloques** encerrándolas entre llaves **{ bloque de sentencias }**.

Cada sentencia puede separarse de la siguiente mediante un **punto y como** (**;**) o un **cambio de linea**. No obstante siempre es recomendable usar el punto y coma como finalizador de una sentencia.

Para ejecutar una función basta con invocarla mediante su nombre y los argumentos que necesite esa función.

Si una variable no existe, se crea en el momento que se usa. Javascript no obliga a declarar o crear la variable antes de usarla. Pero es recomendable. Una misma variable puede almacenar cualquier tipo de datos.

Las **variables** pueden crearse (declararse) mediante las palabras claves **var** y **let**. La primera hace que la variable sea visible dentro de toda la función o el script donde se crea, mientras que let hace que la variable exista solo dentro del bloque de sentencias donde se declare.

Javascript admite la creación de **constantes**. Se usa la palabra clave **const** seguida del nombre de la constante y una asignación de valor.

Operadores

Operadores aritméticos

Asignación =

Este operador significa asignar, guardar o almacenar.

Ejemplo:

```
var lista = 5;
var suma = 1+2;
var nombre = "Juan";
var total = sumando*4;
```

Suma +

Es un operador usado para **sumar** dos valores numéricos o para **concatenar** cadenas entre sí o números y cadenas.

Ejemplo:

```
var cotiza = 40, imp = 5;
var moneda = “$”;
var saludo = "Hola";
var nombre = " Juan";
console.log(saludo + nombre);       /* resultado       Hola Juan*/
console.log(cotiz + imp);       /* resultado:45 */
console.log( cotiza + moneda);        /*resultado: 40$*/
```

Resta -

Operador usado para restar valores numéricos. Puede actuar sobre un único operando numérico cambiándole de signo.

Ejemplo:

```
var num1 = 5, num2 = 4, res = 0;
res = num1 - num2;     /*res contiene 1 */
res = -res;                /*ahora res contiene -1*/
```

Producto (*) y cociente (/)

Realizan las operaciones aritméticas de **multiplicar** y **dividir** dos valores numéricos. En la división si el divisor es 0 no salta ningún error, sino que Javascript lo evalúa dando como resultado la constante **Infinity. Si usas el operador ** se toma como elevado a.**

```
var invitados = 10;
var platos, precio = 20, total;
var base = 2; potencia;
var error;
total = invitados * precio;        /*total: 200 */
platos = total/invitados;          /*platos: 20 */
error = total/0;                   /* Infinity */
potencia = 2**3;                   /* 8, e elevado a 3 */
```

Resto %

También llamado operador módulo, calcula el resto de una división entera.

Ejemplo:

```
var multiplo = 50, divisor = 4, resto;
resto = multiplo % divisor;          /*resto contiene 2 */
```

Incremento (++) y decremento (–)

Estos operadores se usan para incrementar o disminuir en 1 el valor de una variable. Si el operador se antepone a la variable la operación de incremento o decremento es prioritaria sobre cualquier otra.

Ejemplo:

```
var contador = 5, cantidad;
cantidad  = ++contador;  //incrementa contador y luego se guarda en cantidad
cantidad  = --contador;  //Disminuye contador y luego se guarda en cantidad
```

<u>Operadores compuestos</u>

Distinto (delete)

Se usa para eliminar.

Ejemplo: (Borrar un dato del array)

```
var puntos = [2, 45, 67, 90, 24];
delete puntos[3];
console.log(puntos + " longitud "+ puntos.length) //Muestra “2, 45, 67, , 12 longitud5
```

new

Operador unario usado para crear objetos a partir de un constructor. Lo veremos en las próximas clases.

Ejemplo:

```
var fecha = new Date();
console.log("Hoy es "+fecha.toLocaleDateString()) // Devuelve “Hoy es 7/2/2024”
```

typeof

Un operador para determinar el tipo de dato del argumento que se le pase. Los tipos de datos que devuelve son los tipos primitivos o predefinidos de Javascript, es decir: number, string, boolean, function,, object.

Ejemplo:

```
var nom = "Juan";
console.log(typeof nom); //Devuelve string
var x = 12345;
console.log(typeof x); //Devuelve  number
```

Operadores binarios

Complementación ~

Complementa, o niega, una cadena binaria conviertiendo los 1 en 0 y los 0 en 1. Por ejemplo el número 38 escrito en sistema binario es 00100110 si le aplicamos este operador se convierte en 11011001, o sea el -39.

Desplazamiento izquierda << o Desplazamiento derecha >>

Desplaza los bits a la izquierda los lugares que se le indique rellenando con ceros por la derecha. Por ejemplo si al 00000100 (4) lo desplazamos 2 a la izquierda tendremos el 00010000 (16). Como ultiplicar 26 po 4 (2 elevado a 2).

```
var num = 4, res;
res = num << 2;
console.log(res); /* num contendrá  16*/
```

```
var num = 16, res;
res = num >> 2;
console.log(res); /* num contendrá  4*/
```

Existen otros operadores binarios que no se verán en este curso como: AND lógico binario &, OR lógico binario | y XOR lógico binario ^.

Operadores lógicos

Los **operadores lógicos** son los operadores usados por Javascript para trabajar con los datos booleanos o lógicos. Vas a encontrar tres clases de operadores

- **NOT**
- **AND**
- **OR**

La siguiente lista detalla todos los operadores lógicos que puedes usar en Javascript.

Negación: !

Es el operador mas simple de todos, se usa para invertir un valor de **true** a **false** o viceversa. También se pude usar junto a los operadores de igualdad (==) e identidad (===) para invertirlos.

```
var cierto = true;
var falso = !cierto;
console.log ( 5 >4);     //true
console.log (! (5>4));   //false
```

AND lógico &&

Se utiliza para concatenar comparaciones, es decir, para comprobar varias condiciones. El resultado sólo será true si todas las comparaciones o condiciones lo son.

Ejemplo:

```
var edad1 = 2, edad2 = 50, edad3 = 25;
var comp;
comp = (edad1 > edad2) && (edad1 < edad3);      //false
comp = (edad1 < edad2) && (edad1 < edad3);       //true
```

OR lógico ||

Como el anterior, sirve para realizar comparaciones compuestas y solo devolverá false cuando todas las comparaciones los sean.

Ejemplo:

```
var edad1 = 2, edad2 = 50, edad3 = 25;
```

```
var comp;
comp = (edad1 > edad2) || (edad1 < edad3);   /*comp toma el valor true */
```