EE. “ANTONIO DE ESCAÑO”

JavaScript

Metodología de la Programación

Brigada CGA Francisco J Díaz Naya

Documento elaborado por el Brigada CGA Pablo Rodríguez Loira

Índice

4. Estructuras de Control.

4.1. Estructuras de selección (alternativa)

4.2. Estructuras repetitivas.

Página intencionadamente en blanco.

1. Creación de un programa.

No existen una serie de normas absolutas en el proceso de creación de un programa, ya que la programación es una actividad creativa, en la que se unen, por un lado, las ***necesidades*** *del problema, las* ***capacidades*** *que nos proporciona el lenguaje empleado por otro, y la* ***pericia e inventiva*** *del programador para resolver dicho problema.*

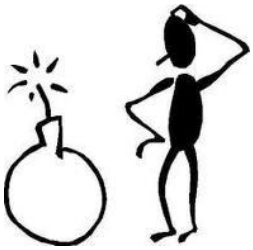
El proceso que sigue desde el planeamiento de un problema, hasta que se tiene una solución instalada en el ordenador lista para ser ejecutada, se compone de varias fases.

- *Definición del problema*
- *Análisis del problema*
- *Diseño de la solución algorítmica*
- *Codificación*
- *Prueba y Depuración*
- *Documentación*
- *Implementación (Producción)*
- *Mantenimiento*

Nota: *Aunque el proceso de crear software es esencialmente un proceso creativo, el seguir esta serie de pasos lógicos conduce a la obtención de programas de mayor calidad. Es muy común que al principio nos saltemos algunos pasos por desconocimiento o pereza, y procedamos directamente a la codificación de los programas. Esta práctica no sólo es incorrecta, sino que hace perder tiempo, dinero y esfuerzo. Aún los programadores experimentados y los profesionales utilizan esta metodología en el desarrollo de sus programas. Los resultados que se obtienen con su aplicación son más confiables, rápidos y seguros que los obtenidos mediante prácticas incorrectas y desordenadas*

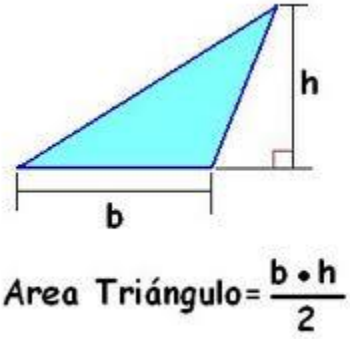
Vamos a explicar cada uno de los puntos, y a la par haremos un ejemplo sencillo "Determinar el área de un triángulo si se conoce la base y la altura." que nos ayudará a asimilar los conceptos.

1.1. Definición del problema.



Consiste en la obtención sin ambigüedades de una visión general y clara del problema. Ayuda a identificar los elementos claves del problema y los de la futura solución, así como fijar los límites de los mismos basados en su planteamiento textual sobre el papel. Un problema mal planteado, incompleto o mal comprendido es un mal inicio para la programación.

Mientras no se comprenda con claridad el problema por resolver, no puede pasarse a la fase siguiente.



En nuestro ejemplo; Debo comprender lo que es el Área y la fórmula que utilizaré.

- ***Definición de Área**: El área de un polígono es la medida de la región o superficie encerrada.*
- ***La fórmula para un triángulo**: Área= (base *altura)/2*

1.2. Análisis del problema.

*En esta fase se definen **la Entrada** que recibirá el programa (datos), **la Salida** que producirá (información) y **el Proceso** necesario para su solución (el método para convertir los datos de entrada en información de salida). A este enfoque se le conoce comúnmente como **E-P-S (Entrada-Proceso-Salida)**.*

En nuestro ejemplo:

Entrada:
La base mediante teclado.
La altura mediante teclado.

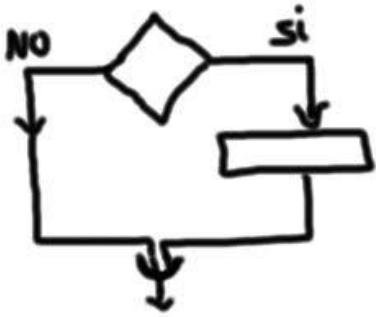
Proceso:
La base la multiplico por la altura y lo divido entre dos.

Salida:
El área presentada en pantalla.

Nota: *En muchos manuales de metodología, fusionan las dos primeras fases en una sola denominada "FASE DE ANALISIS*

Diseño de la solución o algorítmica.

En esta fase se diseña la lógica de la solución a usar, o sea, cómo hará el programa la tarea que se desea automatizar usando los datos de entrada para generar la información de salida. Pueden plantearse diferentes alternativas de solución al problema, debiendo elegir la más adecuada, la que produzca los resultados esperados en el menor tiempo y al menor costo.



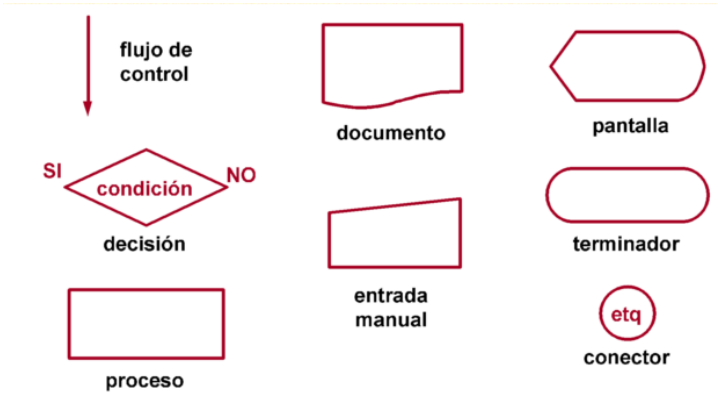
*Un **algoritmo** es una secuencia lógica y cronológica de pasos encaminados a resolver un problema.*

Las acciones básicas que puede llevar a cabo un algoritmo son: pedir datos, desplegar datos, evaluar condiciones y ejecutar operaciones.

*Los programas se estructuran a partir de los **algoritmos**, los cuales se pueden escribir utilizando:*

- ***Pseudocódigo** (mezcla de lenguaje común, términos técnicos de computación, símbolos y palabras reservadas de algún lenguaje de programación)*
- ***Diagramas de flujo** (flujo gramas) que son la representación gráfica de un algoritmo, plasmados en papel para su estudio.*

En la siguiente imagen vemos los símbolos más comunes del diagrama de flujo.



***Nota:** los diagramas de flujo apenas se utilizan, quedando solo su uso para aplicaciones muy sencillas.*

Características de los Algoritmos

- ***Debe tener un punto de inicio** o partida.*
- ***Debe ser preciso** e indicar el orden de realización de cada paso.*
- ***Debe estar bien definido**. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.*
- ***Debe ser finito** (tener un número finito de pasos). Si se sigue un algoritmo, se debe terminar en algún momento.*

*La definición de un algoritmo debe describir con claridad las tres partes fundamentales del problema: **Entrada**, **Proceso** y **Salida** encontrados en la fase anterior.*

*Todos los programas debieran comenzar con un algoritmo en papel, no codificándolo directamente en el ordenador. Aún los programadores más experimentados plasman en papel sus ideas y soluciones antes de programarlas. Pero es común que al principio vallamos directamente al ordenador sin haber siquiera leído bien el problema o pensado siquiera el algoritmo. El culpable de esto, es pensar que no es necesaria la fase algorítmica ya que "los programas al final funcionan igual", pero sólo después de probar diferentes ideas, hacer miles de cambios y perder gran cantidad de tiempo y esfuerzo. Los que se toman tiempo para analizar el problema, pensar y plasmar su solución en papel mediante un algoritmo tendrán un tiempo de respuesta (el tiempo para obtener el programa terminado) mucho menor, y se convierten en mejores programadores. **Es un hecho**.*

En nuestro ejemplo utilizamos un algoritmo en pseudocódigo:

Inicio.
Solicito en la base del teclado.
Solicito la altura del teclado.
*Calcular área=(base * altura)/2*
Mostrar en pantalla el área.
Fin.

1.3. Codificación.

En este paso se traduce el "algoritmo" (diseñado en la fase anterior) al lenguaje de programación que vayamos a utilizar. Convertimos las acciones del algoritmo en instrucciones que el ordenador es capaz de comprender, usando la sintaxis de un lenguaje particular (visual Basic, javascript, java, php...).

Finalizada la escritura del programa, procederemos a su ***compilación****, obteniendo un archivo ejecutable con las instrucciones en un formato denominado código binario, ya comprensible por el ordenador (a través del sistema operativo) para su ejecución.*

Antes de seguir con las fases *vemos brevemente los distintos tipos de lenguaje de programación, los cuales se clasifican en:* ***Lenguaje máquina****,* ***Lenguajes de bajo nivel*** *y* ***Lenguajes de alto nivel****.*

Lenguaje máquina*: Son aquellos que están escritos en lenguajes directamente inteligibles por el ordenador, sus instrucciones son cadenas binarias (0 y 1) que especifican una operación y las posiciones (dirección) de memoria implicadas en la operación.*

Las instrucciones en lenguaje máquina ***dependen de la arquitectura del*** *ordenador y por lo tanto, el lenguaje máquina de un ordenador HP9000 (Hewlett Packard) es diferente que el de 6000 de IBM. Esto con su dificultad para programar es su principal inconveniente.*

Las ventajas de programar en el lenguaje máquina, supone un aumento en la velocidad de ejecución del programa muy superior a cualquier otro lenguaje de programación.

¿No sería mejor programar siempre en lenguaje máquina?

Como respuesta, y a pesar de la mencionada ventaja, podemos decir que existen inconvenientes muy importantes:

- *La vinculación del lenguaje máquina con el hardware del equipo, hace que un programa no pueda ser transportable a otro tipo de ordenador, por lo que debemos rescribir el programa, si necesitamos que sea ejecutado en una máquina con diferente arquitectura que para la que originalmente fue diseñado.*

- *Por otro lado, la escritura de un programa en lenguaje máquina es una labor lenta y compleja.*

***Lenguajes de bajo nivel**: Son más fáciles de utilizar que el "lenguaje máquina", pero también depende del hardware del equipo. El lenguaje de bajo nivel por excelencia es el **ensamblador**. Las instrucciones de este lenguaje son conocidas como nemotécnicos.*

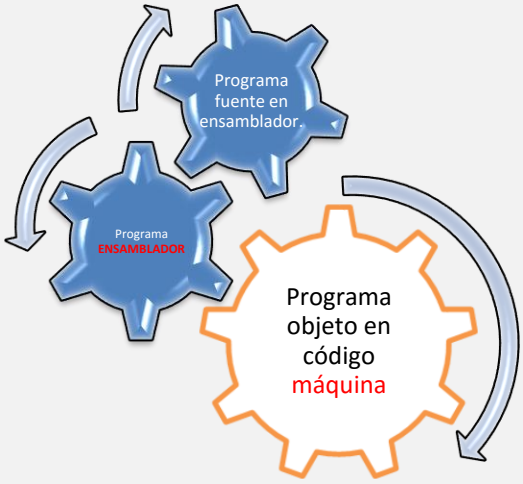
Inglés	*Español*
ADD	*SUM*
SUB	*RES*
DIV	*DIV*

Una instrucción típica de una suma sería:

ADD M, N, P	*Sumar el contenido en la posición de memoria M al número almacenado en la posición de memoria N y situar el resultado en la posición de memoria P*

Un programa escrito en el lenguaje ensamblador no puede ser ejecutado directamente por el ordenador (al contrario que en el lenguaje máquina). Requiere una fase de traducción al lenguaje máquina.
El lenguaje original escrito en lenguaje ensamblador se denomina programa fuente y el lenguaje traducido a lenguaje máquina se conoce como programa objeto.

El traductor que convierte el programa fuente a objeto se llama ENSAMBLADOR igual que el lenguaje (Debido a una mala traducción del inglés al español)



Los inconvenientes del lenguaje ensamblador son:

- *Continúa el problema de la dependencia del microprocesador, lo que impide la transportabilidad de los programas generados con este lenguaje.*
- *La creación de los programas sigue siendo complicada.*

La ventaja del lenguaje ensamblador con respecto al lenguaje máquina es que es más fácil su codificación (más rápido).

Hoy en día los lenguajes ensambladores tienen sus aplicaciones muy reducidas en la programación de aplicaciones y se centran en aplicaciones en tiempo real (sistema de combate de las fragatas…), control de procesos, dispositivos electrónicos…)

***Lenguajes de alto nivel**: Son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en un lenguaje de alto nivel es independiente del hardware de la máquina (no importa en qué ordenador se ejecute).*

Al igual que lo que ocurre con los lenguajes ensambladores, los programas fuente deben ser traducidos por los programas traductores llamados "compiladores". Los lenguajes de alto nivel son muy numerosos aunque la práctica demuestra que su uso mayoritario se reduce a; C, C++, C#, Visual basic.net, Java, JavaScript y Pascal.

Las principales ventajas de los lenguajes de alto nivel son:

- *Tiempo de aprendizaje es menor que los otros lenguajes.*
- *La escritura de los programas es similar al lenguaje de los seres humanos.*
- *Las modificaciones que vayan surgiendo son más fáciles.*
- *Los programas son más baratos.*
- *No dependen del hardware de la máquina, o sea son portables.*

Los inconvenientes son:

- *No aprovecha los recursos internos de la máquina, que se explotan muncho mejor en lenguajes máquina y ensamblador.*
- *Ocupan más memoria.*
- *El tiempo de ejecución de los programas es algo mayor.*

A parte de los lenguajes de programación existe lo que se conocen como los ***Modelos de programación,*** *que son las técnicas empleadas para escribir el código de un programa, de forma que consigamos la mayor efectividad y optimización al ejecutarlo. El desarrollo de las herramientas informáticas ha posibilitado una progresiva evolución en las técnicas de programación, con la consiguiente mejora que ello supone, tanto a la hora de la escritura como de la ejecución de un programa. A continuación vemos los modelos de programación existentes.*

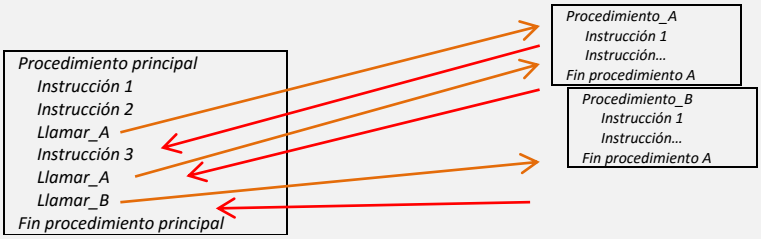
Programación lineal: *(La más tradicional). El código se organiza linealmente, y se ejecuta desde la primera línea a la última de forma consecutiva (no se ejecuta la instrucción 2 si previamente no se ha ejecutado la instrucción 1).*

```
Instrucción 1
Instrucción 2
    ……..
Instrucción n
```

El inconveniente con este modelo de programación, radica en que si dentro de un algoritmo hay un código que necesita repetirse un número determinado de veces, se tendrá que repetir la escritura las veces que sean necesarias.

Programación procedural: *Para solventar los problemas derivados de la programación lineal apareció la programación procedural, que se basa en tomar el código de un programa y dividirlo en partes con entidad propia, denominadas procedimientos o rutinas de código.*

Cada procedimiento se ocupa de resolver una tarea específica, de modo que partiendo de un procedimiento inicial o de nivel superior, se llamará a otro procedimiento para que se ejecute una sub-tarea, y al finalizar devolverá el control o flujo de ejecución al procedimiento que hizo la llamada.



Observa que podemos llamar tantas veces a un subproceso como queramos, con lo que nos ahorramos tener que teclear muchas veces el mismo código.

Programación estructurada: *Apareció como un complemento a la programación procedural, consistente en un modelo de programación que utiliza un conjunto de estructuras de control (condicionales y repetitivas), que simplifican la escritura del código y disminuyen el número de errores del programa.*

A efectos de comprensión imagínate que quiero ejecutar un determinado código si un número es par y otro si es impar. Con las técnicas de programación anteriores no disponemos de un elemento que nos permita hacerlo. Sin embargo con la programación estructurada, podemos utilizar una "estructura condicional" que en pseudocódigo podemos llamar ***si…Fin si*** *(hay otras)*

```
Inicio
Leer número
   Si número es par
         Instrucción…n
   Fin si
   Si número es impar
         Instrucción…n
   Fin si
```

Otro tipo de estructura de control es la repetitiva, y se utiliza para repetir un determinado código un número de veces. Imagínate que quieres escribir tu nombre en pantalla 15000 veces. Con esta estructura ganamos tiempo de escritura.

```
Inicio
Cuenta vale 0
Mientras cuenta < 1000
    Escribe "pablo"
    Aumenta 1 a cuenta
Fin mientras
Fin
```

Programación orientada a objeto (OOP): *Con las clases obtenemos objetos que a su vez contienen miembros. Veámoslo con un símil…, pensar en los planos de un coche, a los cuales le llamaremos* ***clases.*** *No podemos conducir con los planos (clase), pero cuando construimos (**instancia**) con ellos lo que obtengo es un coche (**objeto**) y ese sí podemos conducirlo. Puedo hacer multitud de coches con los mismos planos. Cada coche le puedo dar unas características individuales, pinto de un color determinado, ruedas, motor, caja de cambio... (**Miembros**).*

*Los **miembros** de una clase los podemos dividir en dos grandes bloques:*

- ***Propiedades:** Implementan las características físicas del objeto (color, tamaño,...).*
- ***Métodos:** Implementan el comportamiento del objeto (motor, caja de cambio,...).*

En resumen:

P.O.O		SIMIL	
CLASE		PLANO	
OBJETO		COCHE	
MIEMBROS		CARACTERÍSTICAS	
PROPIEDADES	MÉTODOS	FÍSICAS	COMPORTAMIENTO

A simple vista y sin llegar a profundizar podemos ver qué:

- *Puedo hacer con unos planos tantos coches como quiera.*
- *Modificando un poco los planos, obtengo otros de forma fácil.*

Es una aproximación. Más adelante tendremos varios capítulos destinados a la programación orientada a objetos… por lo que paciencia.

Una vez escrito el programa a partir del algoritmo, en el lenguaje de programación de nuestra elección, necesitamos obtener el archivo ejecutable (.exe) correspondiente.*

El archivo ejecutable y su proceso de creación.

El archivo ejecutable es aquel que necesita el ordenador para hacer funcionar el programa. Un ejecutable (en Windows) es generalmente un archivo con extensión EXE, que obtenemos tras una serie de pasos, que a grandes rasgos describimos a continuación:

- ***Compilación.** Consiste en convertir el código escrito por el programador (código fuente) en el llamado **código objeto.***

***Nota:** Dependiendo de la herramienta de programación utilizada, el código objeto se almacenará en forma de archivo, o será pasado directamente a ejecutable*

Para transformar el código fuente a código objeto se emplea un compilador, que es un programa diseñado para tal fin.

Fig. Proceso de compilación del código del programa.

- ***Enlace.*** *En esta fase se toman, por un lado, el código objeto generado por el compilador, y por el otro, el código de subprogramas de uso general que necesita el lenguaje de programación, que se encuentra habitualmente organizado en archivos denominados librerías o bibliotecas de código (*.dll).*

Todo ello, mediante un programa o utilidad denominado enlazador, se une para obtener el ejecutable final, un archivo con extensión exe. Ver siguiente figura.

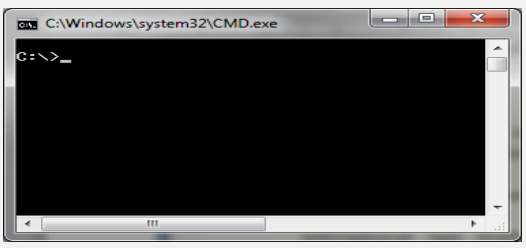
Fig. Proceso de enlace del código objeto y librerías para el ejecutable.

Los usuarios acceden a la aplicación, a través de las interfaces de usuario (lo que ve el usuario en pantalla).

Interfaces de usuario.

Un interfaz de usuario es el aspecto que un programa muestra, y el modo en que interactúa con el usuario que lo ejecuta.

- ***Consola.*** *Un interfaz de usuario de consola consiste en una ventana de tipo MS-DOS o modo de comando, mediante la cual, el usuario introduce las instrucciones en el indicador de comandos del sistema. La creación de programas de tipo consola es sencilla y rápida, y debido a que el objetivo principal de este texto es el aprendizaje de la metodología de la programación, los ejemplos realizados se harán utilizando la consola de comandos*



- ***Escritorio*** *Este tipo de interfaz está basado en el entorno de ventanas del sistema operativo, permitiéndonos crear aplicaciones basadas en formularios o ventanas, con todas las características típicasde este tipo de programas. Por ejemplo el Word, paint…*

- ***Navegador web.*** *Las aplicaciones hechas en JavaScript, php, asp.net… se visualizarán en el navegador web (internet Explorer, Firefox…). Este tipo de aplicaciones no tiene ejecutables (.exe) ya que el código fuente introduce dentro del código HTML bien sea directamente o indirectamente.*

Con la idea de crear los programas más eficientemente disponemos de unas herramientas llamados IDE´S.

Herramientas para la creación de programas.

Remontándonos a los tiempos de las aplicaciones en modo de texto, bajo MS-DOS, con los lenguajes y herramientas de programación disponibles en aquella época, los pasos para la creación de un ejecutable (descritos anteriormente), debían ser realizados por el programador de forma manual, ya que todos los elementos utilizados para la creación del programa se encontraban separados. Disponíamos por un lado, de un programa que consistía en un editor de código fuente (cualquier editor de texto, Word…) por otro lado el compilador, enlazador, librerías, etc.

*Sin embargo, la llegada y evolución de los entornos de desarrollo integrados (**IDE** **I**ntegrated **D**evelopment **E**nvironment), que incorporan en una sola aplicación todos los elementos necesarios para el desarrollo de programas, estas tareas se han automatizado hasta tal nivel, que el programador sólo necesita escribir el código del programa, mientras que los pasos de compilación, enlazado y generación del ejecutable las realiza internamente el propio entorno.*

Nota: *Nosotros no emplearemos IDE para el desarrollo de nuestro código. Emplearemos el "bloc de notas"*

1.4. Prueba y depuración.

Una vez compilado el programa, este es sometido a pruebas a fin de determinar si resuelve o no el problema planteado de forma satisfactoria. Para ello le suministramos datos de prueba, como lo hicimos en la prueba de escritorio. El programa codificado y compilado no garantiza que funcione correctamente, para conseguirlo debemos depurarlo (librarlo de errores de lógica o de ejecución) realizando pruebas continuas con datos y respuestas, verificando todas las posibles alternativas del programa y sus respuestas a fin de determinar si resuelve o no el problema planteado en forma satisfactoria.

Las pruebas que se aplican al programa son de diversa índole y generalmente dependen del tipo de problema que se está resolviendo. Comúnmente se inicia la prueba de un programa introduciendo datos válidos, inválidos e incongruentes y observando cómo reacciona en cada ocasión.

Los resultados obtenidos en las pruebas pueden ser cualquiera de los siguientes:

a. *La lógica del programa está bien, pero hay errores sencillos, los cuales los corregimos eliminando o modificando algunas instrucciones o incluyendo nuevas.*
b. *Hay errores ocasionados por fallas en la lógica, lo que nos obliga a regresar a las fases de Diseño y Codificación para revisión y modificación del diagrama.*
c. *Hay errores muy graves y lo más aconsejable es que regresemos a la fase 2 para analizar nuevamente el problema, y repetir todo el proceso.*
d. *No hay errores y los resultados son los esperados. En este caso guardamos el programa permanentemente en un medio de almacenamiento.*

Puede ser necesario en la mayoría de los casos retroceder a fases previas de desarrollo, revisar el algoritmo otra vez, realizar ajustes al código y una serie de nuevas ejecuciones de prueba para que el programa funcione correctamente.

1.5. Documentación.

Es la fase más ignorada por la mayoría de los programadores, por razones de tiempo, costos o simple pereza. Pero no documentar los programas es un mal hábito en programación y un gran error. Será muy difícil para los usuarios entender un programa si no contamos con un "manual de Usuario". También los programadores que necesiten mantener la aplicación (hacer modificaciones…) necesitan documentación acerca de sus fases de desarrollo.

La documentación se divide en tres partes:

- *Documentación Interna*
- *Documentación Externa*
- *Manual del Usuario*

***Documentación Interna**: Son los comentarios que se añaden al código fuente para clarificarlo.*

***Documentación Externa**: Es todo el material creado y empleado en las diferentes fases del desarrollo del programa. Incluye:*

- *Descripción del Problema*
- *Narrativo con la descripción de la solución*
- *Autor(s)*
- *Algoritmo (diagrama de flujo y/o pseudocódigo)*
- *Código Fuente (programa)*
- *Relación de los elementos utilizados en el programa, cada uno con su respectiva función*
- *Limitaciones del programa*

***Manual del Usuario**: Describe paso a paso la manera cómo funciona el programa, con el fin de que los usuarios pueda operarlo correctamente y obtener los resultados deseados.*

1.6. Implementación.

El programa ya probado, revisado y mejorado se considera terminado y puede utilizarse.

1.7. Mantenimiento.

Es posible que el programa deba revisarse cada cierto tiempo para ajustes. Estos cambios pueden ser por la dinámica del problema, por la naturaleza del código, las exigencias del tiempo o las modernas necesidades que surgen frecuentemente, por lo que se considera que ningún programa es estático. Los programas siempre son susceptibles de mejoras y de mantenimiento. Por tales razones, es común que se tenga que retornar a una de las fases iniciales de desarrollo para corregir o añadir funcionalidades, repitiendo el proceso en cada fase subsiguiente para introducir los cambios pertinentes y lograr que el programa funcione correctamente con los cambios realizados.

Resalto el hecho de que cualquier actualización o cambio en el programa deberá reflejarse en su documentación para que ésta mantenga su vigencia.

Preguntas de repaso.

El proceso que se sigue desde el planeamiento de un problema, hasta que se tiene el programa finalizado, se componen de varias fases. Dime las cuatro primeras fases.

¿Qué fase obtiene sin ambigüedades una visión general del problema y se obtienen los elementos claves del mismo?

En la fase "Análisis del problema", se definen la _____ que recibirá los datos, la _____ que producirá información y el _____necesario para su solución. A este enfoque se le conoce comúnmente como _____

¿En qué fase se diseña la lógica de la solución a usar?

Define lo que es un algoritmo.

Los algoritmos, se pueden escribir utilizando: _____ y _____

En la fase de _____ se traduce el algoritmo al lenguaje de programación que vayamos a utilizar.

Los distintos tipos de lenguajes de programación, se clasifican en: Máquina, _____ y _____.

JavaScript es un lenguaje de "alto nivel", los cuales tienen como principalmente las siguientes ventajas con respecto a los otros; _____ y _____

Los "Modelos de programación" son técnicas que empleamos para escribir código, los cuales son: lineal, _____, _____ y programación orientada a objeto.

¿Qué es el código fuente?

¿Qué es el código objeto?

¿Qué es el código ejecutable?

Un interface de usuario es el aspecto que un programa muestra, y el modo en que interactúa con el usuario que lo ejecuta. ¿Cuáles son las tres "interface de usuario" más comunes?

¿Qué es el IDE?

¿Qué importancia tiene la fase "Prueba y depuración"?

Página intencionadamente en blanco

2. Introducción a JavaScript

*Se supone que debes tener unos conocimientos muy básicos de HTML, para poder seguir este manual. Al ser problemas en su mayoría simples y matemáticos, trataremos en este capítulo la "**fase de programación**" (ver capítulo anterior).*

***JavaScript**, es un lenguaje de programación creado por la empresa "Netscape" con la idea de extender las capacidades del lenguaje HTML. Gracias a JavaScript podemos hacer programas, que se ejecuten directamente en el "navegador" (Explorer, Firefox,…) de manera que nosotros lo vamos a utilizar como lenguaje de iniciación a la programación.*

*Hay quien piensa que JavaScript es lo mismo que java, pero no es así, comparte parte de la sintaxis pero son dos lenguajes similares pero distintos. La principal diferencia entre JavaScript y Java, es que este último es un lenguaje de programación completo y JavaScript es un lenguaje que se integra directamente en páginas HTML. Java se integra en la página HTML mediante **applets** (ver nota), que se colocan en una página Web para llamar a un programa escrito en java.*

***Nota:** Un **applet** es un componente de una aplicación que se ejecuta desde otro programa (en nuestro caso un navegador). Ejemplos comunes son las java **applets**, animaciones Flash*

En sus comienzos se denominó LiveScript, pero justo antes de su lanzamiento se decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de internet de la época.

La primera versión de JavaScript fue un éxito y Netscape Navigator 3.0 (navegador) ya lo incorporaba. Al mismo tiempo, Microsoft lanzó JScript con su navegador Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales. Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript (1997).

JavaScript ha sido la base para lograr una "animación" en la Web sin que se tengan que estudiar lenguajes difíciles de programar y junto con las hojas de estilos (CSS), ha dado lugar a la tecnología denominada Dynamic HTML (DHTML).

Nota: *Antes de entrar en materia debemos decir que utilizando JavaScript es muy conveniente escribir los* ***scripts*** *(código "fuente") a mano sobre el bloc de notas, la cual es la mejor forma de comprender y dominar la sintaxis de este lenguaje*

JavaScript tiene como características principales las siguientes:

- ***Es interpretado*** *no se compila en el navegador, es decir, el programa fuente se ejecuta directamente, con lo que al contrario que los lenguajes compilados no genera "código objeto" ni ejecutable (exe) (ver capitulo anterior).*
- ***El código*** *se integra en las páginas HTML, incluido en las propias páginas.*

2.1. JavaScript en un documento HTML.

Sabemos que un programa en JavaScript no debe ser compilado, ya que el navegador se encarga de leerlo cuando carga la página, pero esto no quiere decir que todo el código en JavaScript se ejecuta nada más cargar la página html, lo normal es que tengamos funciones (subprogramas) hechas también en JavaScript que se ejecuten cuando sean llamados desde el programa principal o desde eventos (pulsar un botón... que haga el usuario desde su navegador).

Tenemos que diferenciar dentro de un documento, lo que es javascript del código html. La inserción en un documento HTML se realiza dentro de la etiqueta HTML ***SCRIPT*** *utilizando la sintaxis:*

```
<SCRIPT type="text / JavaScript [SCR=url]">
Código de script
</script>
```

Los atributos de la etiqueta <script> son:

- ***Type="text/JavaScript"*** *Indica al navegador el lenguaje script utilizado. Nosotros utilizamos javascript pero podría ser: visual Basic script,... Si no pusiéramos nada el navegador asume que es javascript.*
- ***SRC=url*** *El script puede estar situado en un archivo externo. En el atributo SRC le asignamos la ruta de este archivo (URL). Este atributo es opcional, ya que el script normalmente se inserta directamente en el documento HTML.*

A continuación enunciaremos algunos puntos a tener en cuenta respecto a la introducción de JavaScript en un documento HTML:

- *El script insertado mediante la etiqueta **SCRIPT** es leído por el navegador tras la visualización de la página HTML.*
- *La inserción del script mediante la etiqueta **SCRIPT** puede colocarse en cualquier lugar del documento HTML pero se recomienda colocarla en la cabecera, es decir, en la zona definida por el **HEAD**.*
- *La URL correspondiente al JavaScript (src=url acuérdate) tiene la extensión **.js**.*

*El lenguaje JavaScript es **case sensitive**, es decir que distingue las mayúsculas de las minúsculas no es lo mismo **CHORIZO** que **chorizo.***

***Nota:** Los scripts no pueden acceder a los archivos del ordenador del usuario (ni en modo lectura ni escritura) y tampoco pueden modificar las preferencias del navegador.*

Vale ya de tanta teoría, vamos a hacer es nuestro primer programa en JavaScript, al finalizar lo comentamos: Ponemos en un archivo de texto (utilizando el "bloc de notas") el siguiente código.

```
<html>
<head>
    <script language="javascript">
        document.write ("hola mundo");
    </script>
</head>
<body> </body>
</html>
```

Observa que hemos puesto el script en el "head" del documento.

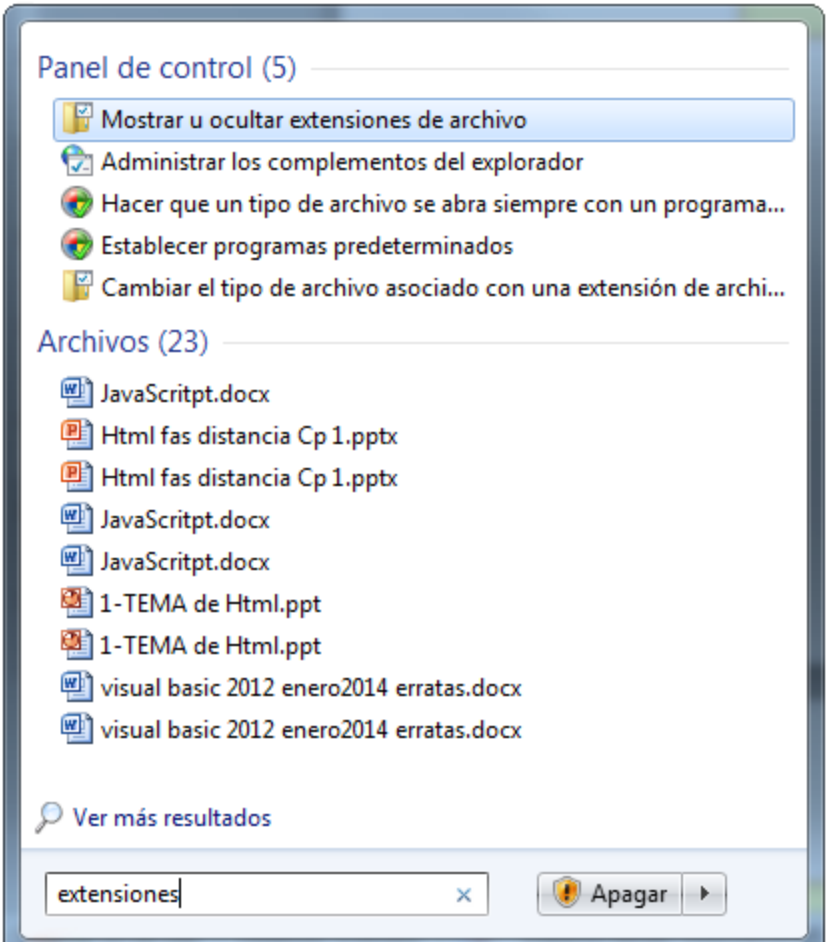
*El código **document.write()**; escribe dentro de la etiqueta <body> lo que coloquemos dentro de los paréntesis (si es texto con comillas simples o dobles). En nuestro ejemplo escribirá el texto "hola mundo"... sigue leyendo, ya que cuando lo ejecutes lo comprobarás por ti mismo.*

Comprueba en el teclado donde se ponen las comillas dobles (“) mediante ***Tecla May + 2*** *y las comillas simples (‘) mediante* ***Tecla ‘ (****donde está el símbolo de cierre de interrogación).*

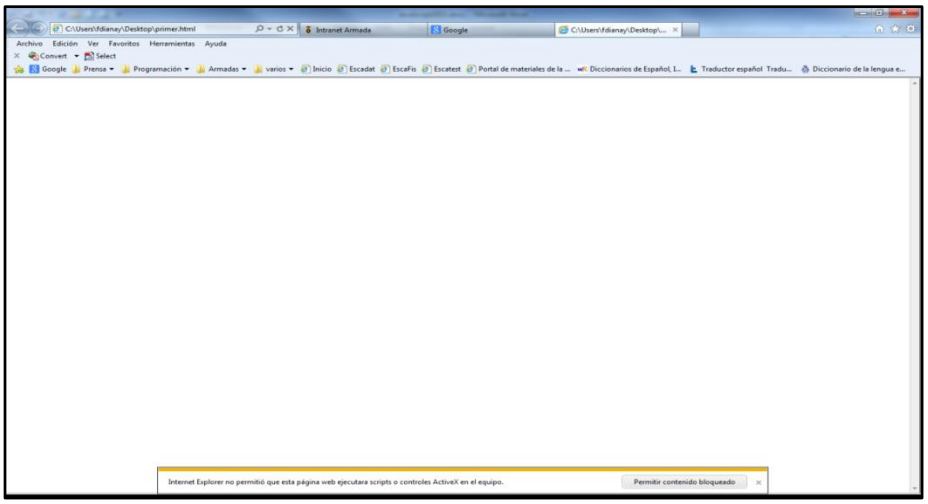
Fíjate en la imagen del teclado:

Guarda el archivo como ***‘primer.txt’****.*

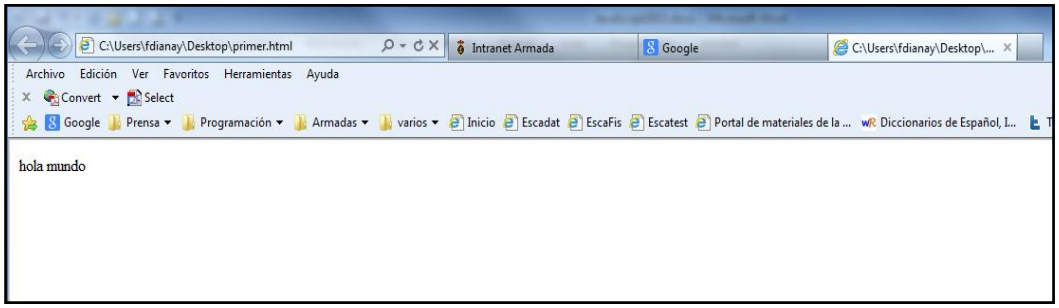
Es importante que tengamos activadas las Herramientas de ***‘ver extensiones’.***



Para poder ver el resultado en el navegador debemos cambiar la extensión de nuestro código de ***.txt*** *a* ***.html*** *y hacemos doble click.*



*Nos aparece una barra de seguridad en el navegador. El navegador ha detectado que hay un script en la página HTML y nos avisa. Para poder acceder al resultado de nuestro script debemos pulsar '****Permitir contenido bloqueado****'.*



Analiza el código anterior, e intenta asimilar que es lo que está pasando.

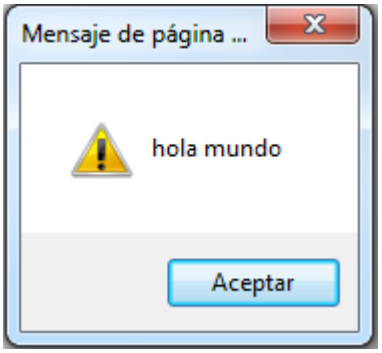
Vamos a ver otra línea de código, para que en vez que saque el texto "hola mundo" en la ventana del navegador (body), lo muestre en una ventana emergente. Para ello sustituimos el código…

document.write ("hola mundo");

Por este otro (nueva):

alert ("hola mundo");

Cuando ejecutamos otra vez el código obtenemos el mismo resultado, pero a través de una ventana emergente.



De momento sabemos dos sentencias Javascript: document.write (“”) y alert (“”). Debes de hacer distintos ejemplos y combinar las dos sentencias antes de pasar al capítulo siguiente.

Preguntas de repaso.

Cuando decimos que el lenguaje JavaScript es interpretado, ¿Quién lo interpreta?

La siguiente afirmación es verdadera o falsa. El atributo src= valor es obligatorio

Donde podemos ubicar el código en JavaScript dentro de un documento html.

La siguiente línea nos muestra una ventana emergente indicando "buenos días". document.write ("buenos días");

Verdadero

Falso

La alerta de "seguridad" que nos aparece cuando ejecutamos un script ¿Qué es lo que hace?

¿En qué etiqueta HTML incluimos el código JAVASCRIPT?

Página intencionadamente en blanco

3. Los Datos y sus tipos.

Todo ordenador maneja datos para convertirlos en información. Estos datos pueden ser las ventas de un supermercado o las calificaciones de una clase.

Los datos se representan indiferentes formas en el ordenador. A nivel de máquina un dato es; un conjunto o secuencia de bits (0 ó 1). Los lenguajes de alto nivel entre ellos JavaScript ignoran los detalles de la representación interna o sea, que yo escribo un 7 y me da absolutamente igual que sea 111 en binario.

Los datos que admiten todos los lenguajes de programación son de tres tipos:

- *Números.*
- *Cadenas de texto.*
- *Booleanos.*

3.1. Tipos numéricos.

No necesitan delimitadores (comillas…). Si necesitamos especificar decimales ***utilizaremos el punto (.)*** *como carácter separador. Con los números podemos realizar* ***operaciones aritméticas*** *las cuales dan como resultado un tipo numérico. Lo dicho tenemos dos tipos de datos numéricos; Reales (decimales) y Enteros. Sigue leyendo… es un poco de teoría y luego retomamos la práctica.*

3.1.1. Tipo numérico entero.

Son los números positivos o negativos no tienen decimales, los cuales se representan exactamente como tenemos por costumbre en el lenguaje común. En programación se utilizan cuatro clases de números, en sus respectivas bases matemáticas.

- ***Decimal:*** *enteros en base 10. 0,1,2,3,4,5,6,7,8,9 alert(10 +);→18*
- ***Hexadecimal****: enteros en base 16. 0, 1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Se coloca antes del número* ***0x****. alert(0x10 + 0x8);→ 24*

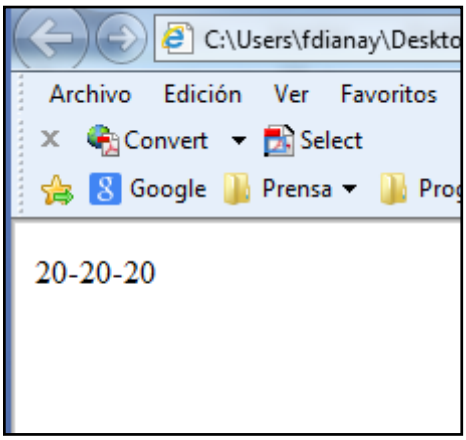
- ***Octal**: enteros en base 8. Colocamos un cero antes de dicho número en octal. 01,02,03,04,05,06,07,08 alert(010 + 08); → 16*
- ***Binario**: En base 2. Son los que utiliza finalmente el corazón de la máquina. Sus dígitos son 0 y 1. No se pueden tratar directamente en javascript.*

Ejemplo: Cuatro tipos de números.

```
20;
0x14            en decimal 20
024             en decimal 20
10100           en decimal 20
```

Si ejecutamos en siguiente código:

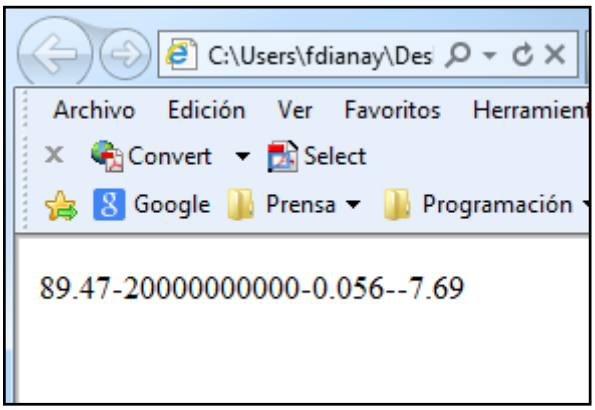
```
<html>
<head>
<script language="javascript">
   document.write (20);
   document.write (-);
   document.write (0x14);
   document.write (-);
   document.write (024);
</script>
</head>
<body></body>
</html>
```



3.1.2. Tipo numérico real.

Los reales se componen de una parte entera y otra fraccionaria separada ***por un punto*** *no por una coma (que no sea por repetirlo). La parte fraccionaria puede estar compuesta por un indicador* ***de exponente E o e seguido de un número entero*** *que indica el valor del exponente. A continuación se mostrarán algunos ejemplos:*

```
<script>
    document.write(89.47);
    document.write("-");
    document.write(2e10);
    document.write("-");
    document.write(5.6e-2);
    document.write("-");
    document.write(-7.69);
</script>
```

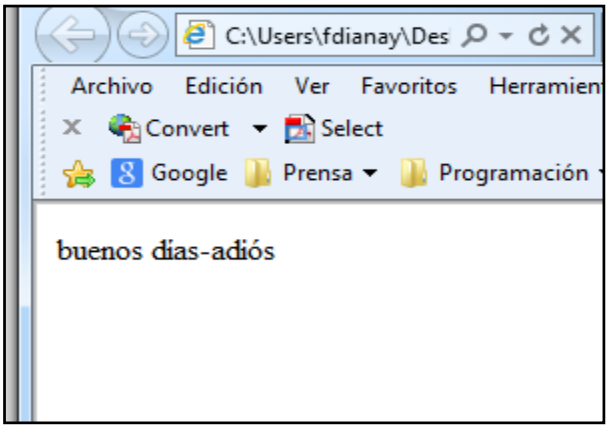


3.2. Tipo cadena (string).

Una ***string*** *es una ristra de caracteres delimitadas por comillas. Las comillas serán simples o dobles atendiendo a una determinada regla. Por ejemplo "chorizo" es una cadena. Puede contener cualquier carácter, incluidos los números (dígitos).*

Nota: *Por defecto se usarán comillas dobles (")*

```
<html>
<head>
<script>
    document.write("buenos días");
    document.write("-");
    document.write('adiós');
</script>
</head>
<body></body>
</html>
```

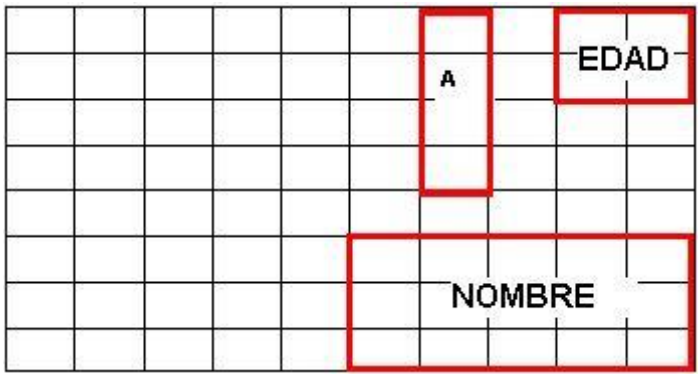


3.3. Las variables

Las variables son trozos de memoria principal (RAM) que se utilizan para guardar datos (números, cadena o booleanos). Los datos almacenados pueden modificarse, o eliminarse cuando deseemos (mientras se ejecuta el programa). Cada variable (trozo de memoria) tiene que tener un nombre para identificarla, el cual es escogido por nosotros pero tiene que respetar una serie de normas:

- *Sólo puede estar formado por letras, números y los **símbolos $ (dólar)** y **_ (guión bajo).***
- *El primer carácter no puede ser un número.*
- *No podemos utilizar palabras reservadas (las que cumplen alguna tarea en el lenguaje) alert, var,…*

Observa la siguiente imagen…, nos simula la memoria RAM, en la que se aprecian tres variables con sus respectivos nombres. El tamaño de cada una de ellas es distinto ya que dependen directamente del tamaño del dato que contengan.



En los apartados siguientes, veremos cómo creamos las variables en la memoria RAM y como le asignamos los datos.

3.3.1. Declarar las variables.

Como ya sabemos cuándo declaramos una variable lo que estoy haciendo es reservar esa porción de memoria, para almacenar un dato. A esa porción de memoria le pongo un nombre. En javascript declaramos una variable utilizando la siguiente sintaxis.

***Var** nombre que se le da a la variable*

En JavaScript no hace falta especificar el tipo de dato que va a almacenar (cadena, número…), ya que se encarga el navegador (internet Explorer…) en función del valor que le pasemos a la variable, de prepararla internamente (modificar el tamaño…) para ese tipo de dato.

Veamos ejemplos de declaración de variables correctas:

```
var x;
var Hola;
var HOLA;
var $numero1;
```

En la ventana anterior vemos cuatro variables declaradas que no tienen asignado ningún valor, fíjate que Hola es distinto de HOLA (te recuerdo que es un lenguaje sensible a las mayúsculas).

Ejercicio: De las siguientes variables indica cuales son válidas:

```
var número;
var 1numero;
var un_mero;
var alert;
```

3.3.2. Asignar o definir las variables.

Definir una variable es asignar un valor (directamente o como una expresión) a una variable.

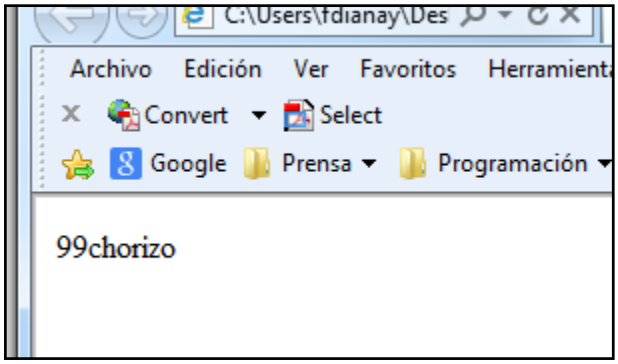
La asignación se puede realizar de muchas maneras pero siempre con el símbolo (=), no debemos confundirlo con el símbolo matemático (igual).

Variable = dato

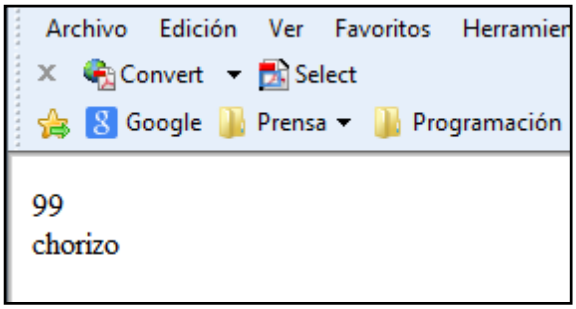
```
x=10; //directamente (le asigno a la variable x el valor diez)

Hola=88 + 12; //como expresión (le asigno a la variable Hola el valor 100)

chorizo="ibérico"; //directamente (le asigno a la variable chorizo el valor "ibérico")
```

*Ejemplo: En el siguiente código podemos ver como se **declara** una variable, que se llama 'chorizo'. Le asigno un tipo de dato numérico (99). A continuación muestro en pantalla el valor que la contiene, haciendo referencia al nombre de la variable, no debes de poner las comillas ya la trataría como un tipo de dato "cadena".*

```
<script language="JavaScript">
  Var chorizo;
  chorizo=99;
   document.write (chorizo);
   document.write ("chorizo");
</script>
```



*Si queremos poner el resultado anterior (99chorizo) en filas distintas acuérdate que podemos utilizar la etiqueta html
* ***como texto****, para que sea interpretada por el navegador, de la siguiente manera.*



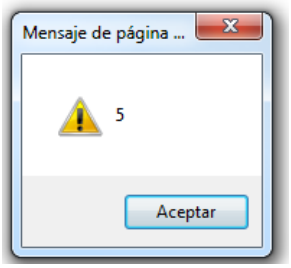

```
<script language="JavaScript">
  Var chorizo;
  chorizo=99;
   document.write (chorizo);
   document.write ("<br>");
   document.write ("chorizo");
</script>
```

Es posible asignar un valor a una variable en el momento de declararla:

Var mi_nombre = valor de la variable

La acción de asignar ***es destructiva****, eso quiere decir que; el valor que tiene en un momento una determinada variable, se pierde cuando se le asigne un nuevo valor. O sea el siguiente dará como resultado* ***5.***

```
<script language="JavaScript"
Var chorizo;
chorizo=99;
chorizo = 5;
alert (chorizo);
</script>
```



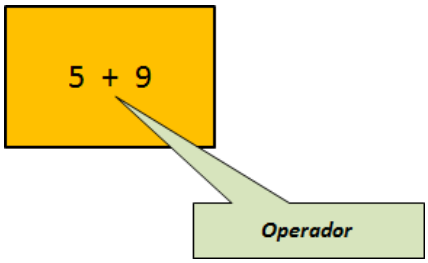
Primero, la variable chorizo vale 99, luego vale 5, quedando este como el valor final.

3.4. Expresiones

Las expresiones son combinaciones de variables/valores dando como resultado un nuevo valor. En función a los tipos de datos que estemos trabajando podremos utilizar diferentes expresiones:

- ***Aritméticas****.- Su resultado es de tipo numérico.*
- ***Cadena****.- Su resultado es de tipo cadena.*
- ***Lógicas****.- Su resultado es de tipo lógico.*

Una expresión consta de operando y operadores.



En función al tipo de expresión a utilizar tendremos que utilizar operadores específicos para dicha expresión.

Nota**: Las expresiones se pueden asignar como dato a una variable. **Por ejemplo: i= 5+2

3.4.1. Expresiones aritméticas.

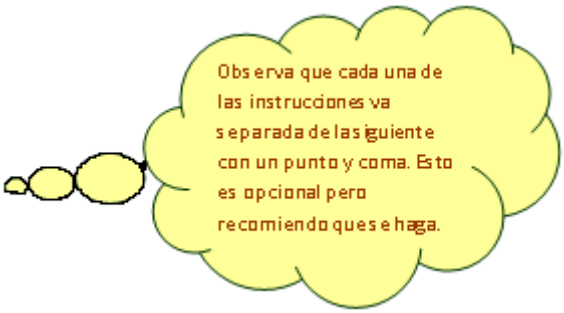
Las expresiones aritméticas son similares a lo que conocemos en "matemáticas". Los datos que utilizan son de tipo numéricos (real o entero).

Los operadores aritméticos toman los valores numéricos (literales o variables) como sus operandos y devuelve un solo valor numérico. Los operadores aritméticos normales son:

Operador	Nombre	Ejemplo	Descripción
+	Suma	5 + 6	Suma dos números
-	Substracción	7 - 9	Resta dos números
*	Multiplicación	6 * 3	Multiplica dos números
/	División	4 / 8	Divide dos números
%	Módulo: el resto después de la división	7 % 2	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
++	Incremento.	a++	Suma 1 al contenido de una variable.
--	Decremento.	a--	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	-a	Invierte el signo de un operando.

Ejemplo1: *Vemos tres variables, y varias operaciones aritméticas. Al final i=14, j=7 y z=7.*

```
var i,
var j;

i=9+5;
j=i/2;
z= j/2
```



Ejercicio para los alumnos*: Genera un código donde declares dos variables que llamaremos a y b. a continuación debemos sacar por pantalla, con un espacio entre ellas, las siguientes operaciones: (a+b); (a-b); (a*b); (b/a) y b--.*

3.4.2. Expresiones Cadena.

Se emplean para los datos tipo cadena y su misión es unir dos o más cadenas en una sola. Solo hay un operador y es ***similar al operador numérico de suma (+).***

OPERADORES ALFANUMÉRICOS	
+	Concatenar

```
<script language="JavaScript">
var nombre = "Soy del";
var nombre2= "Celta";
nombre = nombre + nombre2;
document.write(nombre);
document.write("<br>");
document.write(nombre2);
</script>
```

```
Soy del Celta
Celta
```

3.4.3. Expresiones lógicas o booleanas.

Son aquellas que dan como resultado un valor booleano (verdadero o falso). Los operandos pueden ser Numéricos, Booleanos y String.

Operadores relacionales: *Son idénticos a los que definen las matemáticas...*

Operador	Descripción
==	" Igual a" devuelve true si los operandos son iguales
===	Estrictamente “igual a”
!=	" No igual a" devuelve true si los operandos no son iguales
¡==	Estrictamente “No igual a”
>	“Mayor que" devuelve true si el operador de la izquierda es mayor que el de la derecha.
>=	“Mayor o igual que " devuelve true si el operador de la izquierda es mayor o igual que el de la derecha.
<	“Menor que" devuelve true si el operador de la izquierda es menor que el de la derecha.
<=	"Menor o igual que" devuelve true si el operador de la izquierda es menor o igual que el de la derecha.

El formato general para las comparaciones es:

Expresión 1 ***operador_relacional*** *Expresión2*

El resultado de la operación será ***true*** *o* ***false.***

Ejercicio para los alumnos*: Escribe un código donde declares tres (3) variables, a las cuales nombrarás A, B y C. a estas variables le darás valor de 8 a la variable A, y valor de 3 a las variables B y C. en pantalla debemos obtener el resultado de las siguientes operaciones relacionales, separadas entre cada una de ellas, siguientes: (a==b); (a !=b);(a <b);(a>b);(a>=b);(a<=b);(3==”3”) y (3===”3”).*

```
<script language="JavaScript">
var I;
var=j;

I=10;
J=20
alert(j>i);
</script>
```

Debemos tener cuidado con el ***operador de igualdad (==)****, ya que es el origen de la mayoría de los errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. El operador == se utiliza para comparar el valor de dos variables, por lo que* ***es muy diferente del operador =, que se utiliza como ya sabemos para asignar un valor a una variable****. En el ejemplo siguiente nos muestra el resultado al comparar dos variables, dando como resultado true si son iguales o false si son distintos.*

```
<script language="javascript">

var a=5;
var b=3+2;
alert (a==b)

</script>
```

Operadores Lógicos.

No te pierdas, todavía seguimos en las "expresiones lógicas o booleanas". Combinan sus operandos de acuerdo con las reglas del ***Álgebra de Boole*** *con el fin de producir un nuevo valor que se convierta en el valor de la expresión.*

Los operadores Lógicos se utilizan para combinar múltiples comparaciones en una expresión condicional (que lio…, es más fácil cuando lo veamos). Un operador lógico toma dos operandos cada uno de los cuales es un valor "true" o "false" y devuelve un valor booleano (true o false), ya que te recuerdo que es utilizada en una "expresión lógica o booleana".

Operador	Descripción
&& (May + Tecla 6)	*"Y " Devuelve true todos los operandos son true.* *(5>3) && (5==3+2) → true* *(5>3) && (5==3+3) → false*
|| (alt gr + Tecla1)	*"O " Devuelve true al menos uno de los operandos tiene valor true.* *(5>3) || (5==3+3) → true* *False || true →true*
!	*"No" Obtiene el valor contrario.* *(! true)-->false* *(! 5>3)-->false* *(! 3==2)-->true*

Bueno, en la práctica espero que quede algo más claro. Fíjate como los operandos de la expresión cuando utilizamos los operadores lógicos (código siguiente), son también expresiones lógicas. Te recuerdo que este tipo de operadores (lógicos) los operandos deben de ser valores booleanos, o expresiones que den como resultado valores booleano… Uf que lio! ☺

```
<script language="javascript">
A=8;
B=3;
C=3;
document.write((a==b) &&(c>b));
document.write((a==b) &&(b==c));
document.write( !(b>=c));
document.write(true &&(b==c));

</script>
```

```
False
True
False
true
```

En el ejemplo anterior los valores de las variables siempre valen el mismo valor, por lo cual el ejercicio siempre dará el mismo resultado. Los datos se asignan a las variables normalmente desde el teclado de nuestro ordenador. El cómo hacerlo lo dejamos para el siguiente capítulo.

Operadores paréntesis.

Se utilizan para anidar expresiones (como en matemáticas), pudiendo estar en cualquier expresión, tal como demuestra la siguiente tabla.

Operador Paréntesis	
()	*Paréntesis*

Orden de evaluación de los operadores.

El orden en que se ejecuten en una expresión determinada es muy importante ya que puede alterar el resultado. En JavaScript el orden de ejecución cuando nos encontramos en una expresión distintos operadores es la siguiente.

ORDEN DE EVALUACIÓN	
()	*Comenzando por los más internos*
+-	*Suma y resta.*
**** / %***	*Multiplicación/división/resto*
< <= >= >	*Relacionales*
== != === !===	*Igualdad*
&&	*And (lógico)*
||	*Or (lógico)*
*Si encontramos operadores del mismo nº orden, se ejecutan primero los de la izquierda. Ejemplo 5/3*2%5 sería ((5/3)*2) %5, por lo que aconsejo que siempre utilicemos los paréntesis.*	

Es un lio y es difícil aprenderlo por lo cual nosotros usaremos los paréntesis para controlar que las operaciones se lleven a cabo según el orden que nosotros queramos.

3.5. Comentarios en el código.

Es uno de los elementos más simples de los que se compone un lenguaje de programación, aunque no por ello son los menos importantes, estamos hablando de los comentarios.
Los comentarios en el código permiten insertar observaciones del autor del código para describir las distintas partes del programa. El intérprete JavaScript los ignora y posee por ello una sintaxis particular.
Se distinguen los comentarios en una sola línea, precedidos por la barra oblicua doble ***//*** *(la barra del siete) y los comentarios en varias líneas delimitados por los símbolos* ***/**** *y por* ****/****.*

Por ejemplo:

```
// Esto es un comentario

/* Esto es un comentario */
```

Los comentarios de código no se tienen en cuenta por el navegador porque su utilidad está limitada a la comprensión o aclaración del código por parte del programador.

Poner un ejemplo de un comentario mediante el uso de cualquier código.

3.6. La conversión de datos.

Antes que nada, recordar que JavaScript no es un lenguaje con tipos; por consiguiente, cuando se declara una variable no se sabe el tipo de dato que contendrá (numérico, cadena,...), solo cuando se asigne el dato (a la variable) es cuando sabe el tipo de dato.

A veces necesitamos convertir un dato de un tipo a otro. Imagínate "25" como cadena de texto quiero convertirlo a 25 para poder tratarlo en una operación aritmética...

El contenido de la variable se convertirá a otro de forma automática en el transcurso del programa según su uso.

Por ejemplo, el resultado de la variable z nos muestra el resultado de una conversión automática correcta solo cuando se utiliza el operador aritmético de multiplicación (*) ya que no hay posibilidad de error. En cambio cuando se utiliza el operador aritmético suma (+) se confunde con el operador de cadena (+), por lo que el resultado lo muestra concatenado… analiza el código y compréndelo.

```
<script language="javascript">
var una_cadena="7";
var un_numero=42;
var x = una_ cadena + un_numero;
var y= un_número + una_cadena;
var z= un_numero * una_cadena;
document.write (x + <br> + y + <br> + z);
</script>
```

```
742
427
294
```

Nota: *Más adelante ya veremos cómo podemos convertir los datos de forma manual ya que la conversión automática es más liosa.*

Preguntas de repaso.

¿Cuáles son los tipos de datos que admiten todos los lenguajes de programación?

El dato 0x88 es un tipo de dato numérico del tipo..... y 8.9 es del tipo.....

Los tipos de datos booleano ¿Cuantos valores puede tener?

Una string está delimitada por.....

Haz un programa que saque por ventana el valor de la variable casa, la cual tiene asignado el valor "azul"

El siguiente código da como resultado…

```
<html>
<head>
        <script language="javascript">
        var i ;
        i=99;
        var zz = i + 33 * 9;
        alert(zz==8 || 5>2);
        </script>
</head>
<body></body>
</html>
```

¿Qué tipo de expresión utiliza el operador de concatenación?

Cuando quiero comentar 10 líneas de código utilizo.....

Página intencionadamente en blanco

4. Estructuras de Control

*Los **scripts** vistos hasta ahora han sido tremendamente sencillos y lineales: se iban ejecutando las distintas líneas de código una detrás de la otra desde el principio hasta el fin ("programación lineal", ver cap. 1). Sin embargo, esto no tiene por qué ser siempre así, ya que en los programas generalmente necesitaremos hacer cosas distintas dependiendo del estado de nuestras variables, o realizar un mismo proceso muchas veces, sin re escribir la misma línea de código una y otra vez. En este capítulo vamos a ver el modelo de programación estructurada (ver capítulo 1), la cual utiliza un conjunto de estructuras de control (selección y repetitivas).*

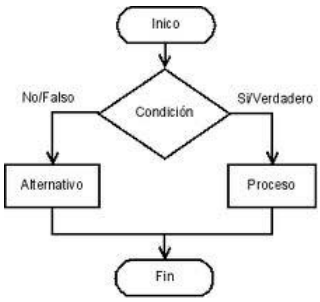
Si se utilizan estructuras de control, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas "inteligentes" que pueden tomar decisiones en función del valor de las variables.

JavaScript posee las sentencias de control típicas de los lenguajes de programación:

- ***Estructura de selección o alternativa**: "Si se cumple una determinada condición hazlo, si no se cumple haz otra cosa".*
- ***Estructuras repetitivas o bucles**: "Repite este código mientras se cumpla una determinada condición".*

4.1 Estructura de selección (alternativa).

En ocasiones nos puede interesar que se ejecute un conjunto de instrucciones, cuando se cumplen una o varias condiciones.



La condición es una expresión "lógica o booleana". O sea si el resultado de la expresión (de la condición) da como resultado 'true' se ejecutará un determinado código, y si da como resultado "falso", se ejecutará otro código distinto.

La estructura de selección típica que tienen todos los lenguajes de programación es la If…

*La estructura de selección **if** tiene la siguiente sintaxis:*

```
If (Condición) {
    Instrucciones 1;
}
Else {
    Instrucciones 2;
}
```

Nota:** Los paréntesis asociados que delimitan la condición **no son opcionales.** En caso de que la condición sea verdadera se cumplirán las Instrucciones 1, en caso contrario se cumplirán las Instrucciones2. El conjunto de instrucciones a ejecutar estarán delimitadas por **corchetes.

Ejemplo básico.

```
<script language="javascript">
Var permitir =true;
If (permitir==true){
Document.write ("Entra");
}
Else{
Document.write("Salir");
}
</script>
```

Entra

En este caso, la condición es una comparación entre el valor de la variable permitir y el valor true. Como los dos valores coinciden, la igualdad se cumple y por lo tanto la condición es cierta, su valor es true y se ejecutan las instrucciones contenidas en ese bloque del if.

La comparación del ejemplo anterior da lugar a muchos errores de programación, al confundir los operadores == y =. Las comparaciones siempre se realizan con el operador ==, ya que el operador = solamente asigna valores.

El siguiente código da error. ¿Sabrías decir por qué?

```
<script language="javascript">
Var permitir =true;
If (permitir==true){
Document.write ("Entra");
}
Else{
Document.write("Salir");
}
</script>
```

Solución problema anterior: *Efectivamente, la condición estamos utilizando el operador de asignación (=) en vez del operador relacional (==).*

La condición del if (condición) puede combinar los diferentes operadores lógicos y relacionales... en el ejemplo siguiente utilizo el operador relacional not (!), por lo que el resultado de la condición (!permitir==true)) da como resultado false, por lo que se ejecuta el bloque del else--> salir.

Ejemplo básico uso de operadores relacionales*. Si en el código anterior la condición fuera* ***(!permitir==true)****, ¿qué resultado nos ofrecerá el navegador?*

Los operadores lógicos AND (&&) y OR (||), nos permiten encadenar varias condiciones simples para construir condiciones complejas...

Ejemplo básico uso de operadores lógicos*. Siguiendo con el código que ya hemos usado ¿qué resultado nos ofrece el código en caso de que la condición sea* ***(permitir==true && nombre ==”Pablo”****? Como ves se ha incluido una nueva variable. Inclúyela en el código y ofrece el resultado.*

Nota: *No es obligatorio poner en el código la palabra clave* ***else****. El código seguirá funcionando igual de bien.*

Ejecuta el siguiente problema quitando la palabra else*. Declara una variable num con un valor asignado de 10. La condición será (num>5). Escribe la condición con Intrucciones1=Verdadero e Instrucciones2=Falso. ¿Cuál será el resultado en pantalla?*

***Analiza el siguiente ejemplo**. Hacer un programa en JavaScript que me diga si la variable **i** es mayor o menor que la variable **j**. El resultado en vez de presentarlo en un mensaje emergente de alerta, escríbelo en el propio documento con la instrucción document.write("")*

```
<script language="javascript")>
var i=50;
var j=20;
    If(i>j){
       document.write ("I es mayor");
    }
   else{
       document.write ("j es mayor");
   }
</script>
```

***Nota:** Es obligatorio que el código este delimitado por llaves y que las variables e instrucciones al final llevan punto y coma (;)*

Para continuar con lo básico del lenguaje Javascript, debemos de tener en cuenta que la asignación de valores a las variables no siempre se pone fija y tienen el mismo valor. La entrada de datos si fuera así siempre sería aburrida y poco práctica.

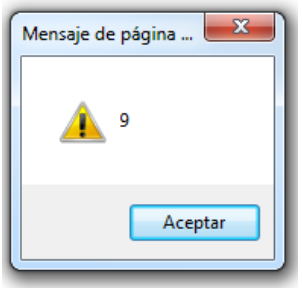
*Existe la posibilidad de introducir datos desde **el teclado** mediante el uso de formularios. Además existe una forma sencilla de introducción de datos desde el teclado mediante el uso del código **prompt().***

Veamos un ejemplo…

```
<script language="javascript")>
var edad;
edad=prompt("Introduce una edad:","");
alert(edad);
</script>
```

En un primer momento aparece una ventana emergente (similar a la del "alert()"), pero con un mensaje y un cuadro de texto para introducir un dato. Observa que coinciden con los parámetros que utilizamos en el prompt("texto","valor predeterminado").

Supongamos que ponemos un valor de 9 y pulsamos "Aceptar"…

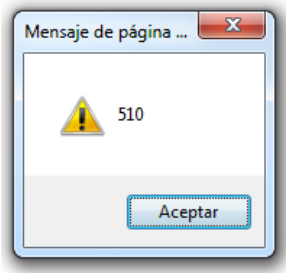


Es importante recordar que mientras no pulsemos Aceptar no se ejecutará el código.

Tenemos un pequeño problema cuando solucionamos el siguiente código…

```
<script languge="javascript">
var num;
num=prompt("Introduce un número:","");
var num2=num+10;
alert(num2);
</script>
```

Introduzco por teclado el número 5 y el resultado es…



¿Dónde crees que puedo haber metido la pata? o, ¿Es correcto el resultado que aparece en el alert?

***Solución cuestión anterior**. Observamos que no hace lo que queremos, te recuerdo que ya lo vimos antes (conversión automática de datos). Pensábamos que daría 15 y da como resultado 510. El problema radica en que **prompt** da como resultado una cadena de texto no un número y como tal lo trata el navegador.*

*Para solucionar "el problema" debemos de utilizar un **"conversor de cadena a número"**. Los cuales funcionarán si el texto que queramos convertir, es convertible a número, por ejemplo; "7" es convertible (7), pero "z" no (?).*

***Tenemos dos conversores**.*

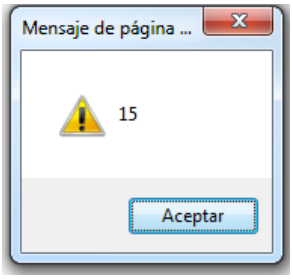
Conversor cadena a número entero:

```
parseInt("5");
```

Conversor cadena a número real (recuerda los decimales):

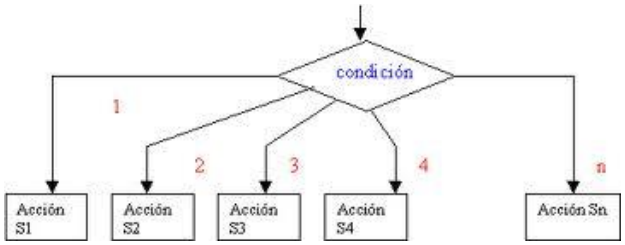
```
parseFloat("3.5")
```

*Si modificamos el código anterior: 'num **parseInt(**prompt("Introduce un número:",""));' o también 'num=parseInt(num);', el resultado será:*



***Ejercicio para los alumnos**. Escribe un código donde se declare una variable edad, la cual cargará los datos desde teclado. El código nos debe proporcionar un mensaje de alerta que nos indique si la persona es mayor de edad o menor de edad.*

A veces es necesario que existan más de dos selecciones posibles, normalmente este tipo de estructura se pueden resolver mediante estructuras de selección simples (if… else). Pero si el número de alternativas es grande dificultará su manejo por lo que es conveniente utilizar las estructuras ***Condicionales Múltiples.***



Tiene la siguiente sintaxis:

```
If (Condición) {
Instrucciones1;
} else if (Condicion2){
Instrucciones2;
} else if (Condición n){
Instrucciones n;
}
```

Lo valores que se introducen se evalúan con la primera condición, y en caso de que no cumpla se evalúa con las siguientes condiciones.

Observa un ejemplo.

```
<script language"javascript">
var dia;
dia =parseInt(prompt("Introduce un número:",""));
If (dia ==1){
document.write("Código Lunes");
} else if (dia ==2){
document.write ("Código Martes");
Introduce así el código hasta 7 que será Domingo. La última condición será:
} else (dia >7){
document.write("No corresponde a ningún día de la semana");}
</script>
```

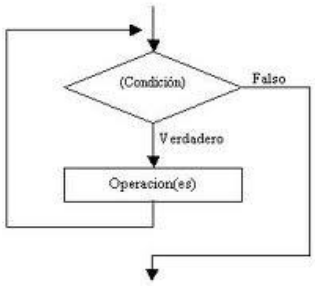
***Ejercicio para los alumnos**. Realice un programa que nos sirva de conversor rápido de las antiguas pesetas a los euros actuales.*

***Ejercicio para los alumnos**. Redacta un código que nos permita introducir un número por teclado y que nos permita comprobar si dicho número es mayor o igual a cero (0) pero menor que cinco (5). En caso de que el número cumpla nuestros requisitos, debemos recibir un mensaje en pantalla en código string (me explico, en caso de que se marque 1, en pantalla aparecerá uno) y si el número no cumple con las condiciones salga una alerta indicando ERROR.*

4.2. La estructura repetitiva.

*Esta estructura (repetitiva) se utilizará cuando necesitemos repetir un determinado código un número determinado de veces (p.e. escribir mi nombre 2000 veces en pantalla) o cuando queremos que se ejecute un código mientras se cumpla una determinada condición (p.e. mientras la variable **contraseña** contenga un valor distinto del valor "chorizo"). La "estructura repetitiva" se conoce comúnmente como bucles o ciclos y las veces que se repiten se llama iteración. En cada iteración se debe de modificar la condición, para que finalice el bucle. Hay varias estructuras, nosotros en este curso solo vemos las estructuras repetitivas ya que son las más ampliamente aceptadas.*

Estructura de repetición WHILE.



Es frecuente que determinados cálculos requieran la ejecución repetitiva de un conjunto de sentencias, mientras ocurre una determinada condición. Lo primero que se hace es evaluar la condición, si no se cumple la condición, no se ejecutan las condiciones, y el programa sigue en la siguiente instrucción, si se cumple, se ejecutan las condiciones, y al final se vuelve a evaluar la condición. El proceso se repite indefinidamente mientras la condición sea cierta.

Tiene como sintaxis:

```
While (Condición) {
    Instrucción o bloque de instrucciones;
    Modificador de la condición;
}
```

Nota: *El modificador de la condición, modifica el valor de la variable de la condición. Por normativa dicha variable suele llamarse i, pero se le puede llamar como uno quiera*

***Observa el siguiente y sencillo código**. Buscamos imprimir los números desde 0 hasta 19.*

```
<script language="javascript">
var i;
i=0;
While (i<20){
document.write(I + " ");
i=i+1;
}
</script>
```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Vamos a ver el funcionamiento, línea a línea.

1.- Declaro una variable que se llama i y le asigno el valor 0.

2.- Comienza la estructura repetitiva

3.- Evalúa la condición.

4.- Escribe en ventana el valor de i

5.- Incrementa en 1 el valor que contiene la variable, con la idea que se modifique el la condición. En el supuesto que no se modifique el valor de i, nunca saldríamos de la estructura repetitiva.

*Si quiero aplicar salto de línea utilizo las etiquetas html, de toda la vida, en este caso
.*

***Ejercicio para los alumnos**. Haz un programa que nos presente en pantalla los primeros números pares.*

***Ejercicio para los alumnos**. Crear un programa que nos verifique contraseñas con un máximo de tres intentos. La pass será 12345. Si pass no es correcto debe emerger una alerta "Password Incorrecto", y si, la pass es correcta aparezca en pantalla "Bienvenido a mi programa".*

Preguntas de repaso.

¿Cuáles son las sentencias de control en javascript?

En la estructura alternativa, que delimita el trozo de código a ejecutar (conjunto de instrucciones), cuando se cumple una determinada condición.

Cuando quiero asignar un valor a una variable sin ser desde código, puede ser a través de un formulario o también con la palabra reservada.....

Crea un código que convierta la cadena de texto "765" en tipo de dato numérico 765

Escribe la sintaxis de la estructura condicional múltiple.

Quiero presentar en el navegador 400 veces "Me gusta el jamón", que tipo de estructura utilizaré. Realiza un programa que lo haga.

La condición en un bucle, delimita las veces que se ejecuta un determinado código. ¿Explica brevemente como hacemos para modificar dicha condición?

Crea un programa que sume los cinco primeros números pares. Utilizando un bucle.

Modifica el programa anterior para que nos muestre los cinco primeros números pares mayores que 2000

Crea un programa que nos muestre los 100 primeros números primos (Sólo es divisible entre sí mismo y 1) (tiene un nivel de dificultad alto, si no te sale pasa al siguiente)

Hacer un programa que imprima 5 veces la serie de números del 0 al 4 (una pista. Un bucle dentro de otro bucle)

Solicitar tres números por teclado y presentarlos de forma ordenada.

Crea un programa que introduzca números por teclado hasta que su suma rebase el 1000. Al final mostrará el valor de la suma y la cantidad de números introducidos.

Introducir por el teclado nos números inferiores a 50. Al más pequeño lo aumentamos de 5 en 5 y al mayor decrementándolo de 2 en 2. Al final mostrar las dos series de forma alternativa hasta que el menor supere al mayor.

En defensa se han vuelto locos y deciden darnos una prima en función de los primos que tenga cada uno.

Si no tiene hijos le corresponde 2 euros.

Si tiene uno o dos hijos 10 euros

Si tienen tres hijos 40 euros

Si tiene entre 4 y 7 (ambos inclusive) le corresponde 200 euros y un bote de colonia.

Si tiene más de 7 dan la transitoria y le ponen de nombre a un buque.

Realiza un programa, que introduciendo el número de hijos, nos diga la prima que nos tocaría (cuando digo prima no me refiero a los hijo de mi tío/a).

Página intencionadamente en blanco