INTRODUCCIÓN A LA PROGRAMACIÓN

Programa – Aplicación - Software

Tres términos que a menudo usamos en forma indistinta, pero que hablando de manera ESTRICTA podemos establecer algunas diferencias.

Programa

Podemos definir un programa como un conjunto de instrucciones que ejecuta un procesador de computadora (Para realizar cálculos, guardar y/o borrar datos el equipo, etc). Todo programa tendrá un conjunto finito de instrucciones, las cuales se van ejecutando 1 a 1 en cadena hasta finalizar la ejecución.

Software

El término "software" hace alusión NO a un único programa; sino a un CONJUNTO de ellos; los cuales pueden tener interacción entre sí. Dentro del “software” incorporamos también posibles bases de datos, archivos y/o librerías que necesitan consultar dicho conjunto de programas.

El software NO necesariamente tiene interacción directa con el usuario: pueden residir en un segundo plano de ejecución y trabajar de manera transparente para él.

Ejemplo: Los llamados “drivers”: software que establecen comunicación con los periféricos y dispositivos (piezas físicas) de un equipo.

Cuando hablamos de "software" nos referimos a programas en general, en cambio cuando hablamos de un software con un propósito específico y concreto, estamos hablamos de una “aplicación”.

Aplicación

Una aplicación es un programa específico, que resuelve un problema concreto. A menudo hablamos de aplicaciones contables, aplicaciones de gestión de RR.HH, aplicaciones de liquidación de sueldos, etc.

Una de las características principales de las aplicaciones es la interacción directa con el usuario: Las primeras aplicaciones tenían interacción con el usuario únicamente a través del teclado (aplicaciones de consola); actualmente disponemos del control del mouse del equipo el cual le otorga mucha más dinámica a la aplicación: Normalmente las mismas disponen de ventanas de diálogo, cajas de escritura, botones, menús, etc.

Por otra parte, las instrucciones a ejecutar no siempre serán disparadas en forma automática: El usuario deberá indicar que y cuando ejecutar una determinada acción.

***Observación**:*

Al procesador de una computadora no le interesa saber con qué tipo de aplicación estamos trabajando, para dicho procesador nuestra aplicación es un mero programa que debe ejecutar.

Sistemas Informáticos

Un sistema está formado por un conjunto de programas, involucra también herramientas hardware (partes físicas: monitores, teclados, impresoras, etc), redes de comunicación, bases de datos, servidores, etc.

Cuando hablamos de sistema informático también involucramos a la parte HUMANA (usuarios finales, personal de soporte, programadores, etc).

Muchas veces hablamos de:

- **Sistemas de seguridad**: Conjunto de cámaras, alarmas, cableados, programas que guardan archivos de video con grabaciones diarias, programas de monitoreo y activación de cámaras, servidores en donde alojamos registros de actividades, personal de monitoreo y técnico, etc.

- **Sistemas de fichaje de empleados**: Tarjetas magnéticas, máquina de fichajes, servidores, programas que reportan entradas y salidas, los mismos empleados, etc.

- **Sistemas bancarios**: Cajeros automáticos, tarjetas de débito / crédito, servidores, bases de datos con información de los clientes, personal bancario, etc.

Y muchos más.

Tipos de aplicaciones

Existen diversos criterios para clasificar a las aplicaciones, veamos 2 de ellos:

Según su ámbito

Aplicaciones de escritorio:

Son aquellas que típicamente corren en un sistema Windows; las cuales pueden ser abiertas yendo a la lista de programas instalados en el sistema operativo. Dichas aplicaciones trabajan con ventanas, tienen un menú en la parte superior (Con opciones tales como: archivos, herramientas, configuración, etc). Estas aplicaciones permiten ingresar datos, obtener reportes de datos, etc. Existe mucha interacción con el teclado y el mouse de la computadora. El botón secundario del mouse nos suele generar el conocido menú contextual, muy útil ya que representa un atajo para la ejecución de una funcionalidad específica.

Podemos decir que las aplicaciones de escritorio son las “aplicaciones tradicionales” de interfaz gráfica.

Aplicaciones de Consola:

Son aquellas aplicaciones que utilizan una ventana de MS-DOS como salida. Estas aplicaciones carecen de interfaz gráfica y son las primeras que aparecieron. Todas las instrucciones del usuario serán dadas únicamente a través del teclado.

Aplicaciones Web:

Son aquellas que son accedidas desde un browser (Internet Explorer, Firefox, Chrome, etc) a través de alguna dirección web (Ej: www.algunlado.com), la cual establecerá una conexión a un servidor en internet. No solo nos referimos a sitios webs; también podemos encontrarnos con aplicaciones de gestión, administración, financieras, etc.). La gran ventaja de este tipo de aplicaciones es que pueden ser accedidas desde cualquier equipo y/o dispositivo, desde cualquier parte del mundo.

Aplicaciones Mobile:

Aquellas que funcionan sobre dispositivos mobile (tablets, celulares, etc). Se trata nada más y nada menos de las famosas "apps"; las cuales han tomado muchísimo auge en los últimos tiempos. Muchas de ellas son descargadas desde los “stores” de aplicaciones del dispositivo.

Según la manera en que resuelven problemas:

Aplicaciones "enlatadas":

Son aquellas que se programan de manera que resuelva determinados problemas en determinada forma. Las aplicaciones de tipo “enlatadas” se crean con el propósito de reparto y venta masiva; muchas veces instalables desde un CD/DVD-R. Las aplicaciones de tipo enlatadas NO tienen en cuenta ningún detalle del contexto del usuario que opera con él (el usuario debe adaptarse a lo que ofrece la aplicación). Normalmente estas aplicaciones se van actualizando con el tiempo de manera de ir evolucionando en versiones (1.0, 2.0, 2.5, etc).

Aplicaciones "hechas a medida":

Son aquellas que se programan para resolver problemas en forma particular y específica acorde a las necesidades del cliente interesado en desarrollarla. Suelen llevar bastante tiempo para crearlas, y requieren mucha interacción con el cliente, de manera de ir ajustando progresivamente distintos parámetros (pantallas, formato de los datos que aparecen en las distintas pantallas, forma de interactuar con el usuario, etc).

El cliente es quien decide como trabajará el sistema, que funcionalidades abarcará, cuál es su alcance, cuáles son sus limitaciones, y como se dispondrán en pantalla los distintos elementos (información, formularios, reportes, botones, etc).

Finalmente destacamos que algunas aplicaciones solo funcionan en determinados sistemas operativos (S.O), tal como Windows, o bien pueden funcionar en distintos S.O (Windows, Unix, Solaris, etc); en este último caso hablamos de aplicaciones MULTIPLATAFORMA.

Las razones de porqué algunas aplicaciones funcionan solo en determinados S.O ó bien en distintos, se encuentran relacionados con la tecnología con la que fue construida dicha aplicación.

Lenguajes de Programación

Se trata de un lenguaje formal, con reglas estrictas de escritura, el cual permite comunicarle a una computadora que es lo que debe hacer con absoluto detalle. Todo lenguaje de programación se conforma por un conjunto de símbolos, signos de puntuación, operadores, valores, palabras clave e identificadores que permiten escribir las instrucciones a ejecutar.

A través de los lenguajes de programación podemos crear nuestros programas.

Existen docenas y docenas de lenguajes de programación hoy día, muchos con similitudes entre sí, como también así con sus diferencias.

Tipos de Lenguajes

Existen muchos criterios para clasificar a los lenguajes de programación, veamos 2 de ellos:

Cercanía al lenguaje máquina.

El lenguaje máquina es el “verdadero lenguaje” que comprende un procesador de computadora: Todo lo programado en un lenguaje “NO máquina” será al final de cuentas traducido a este lenguaje. Este lenguaje solamente trabaja con valores 1 y 0 (a menudo lo llamamos: “lenguaje binario”). Dentro de esta clasificación tenemos lenguajes de "bajo nivel" y de "alto nivel".

- **Lenguajes de BAJO Nivel:**

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A3  ES=17A3  SS=17A3  CS=17A3  IP=0100   NV UP EI PL NZ NA PO NC
17A3:0100 0000          ADD     [BX+SI],AL                          DS:0000=CD
-a
17A3:0100 mov ah,8
17A3:0102 add ah,3
17A3:0105 sub ah,4
17A3:0108 int 20
17A3:010A
-t

AX=0800  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A3  ES=17A3  SS=17A3  CS=17A3  IP=0102   NV UP EI PL NZ NA PO NC
17A3:0102 80C403        ADD     AH,03
-t

AX=0B00  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A3  ES=17A3  SS=17A3  CS=17A3  IP=0105   NV UP EI PL NZ NA PO NC
17A3:0105 80EC04        SUB     AH,04
-t

AX=0700  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A3  ES=17A3  SS=17A3  CS=17A3  IP=0108   NV UP EI PL NZ NA PO NC
17A3:0108 CD20          INT     20
-
```

Son aquellos que se aproximan bastante al lenguaje máquina: son lenguajes complejos de utilizar y comprender. Uno de los más populares: Assembler, el cual es muy utilizado en el campo de la electrónica.

- **Lenguajes de ALTO Nivel:**

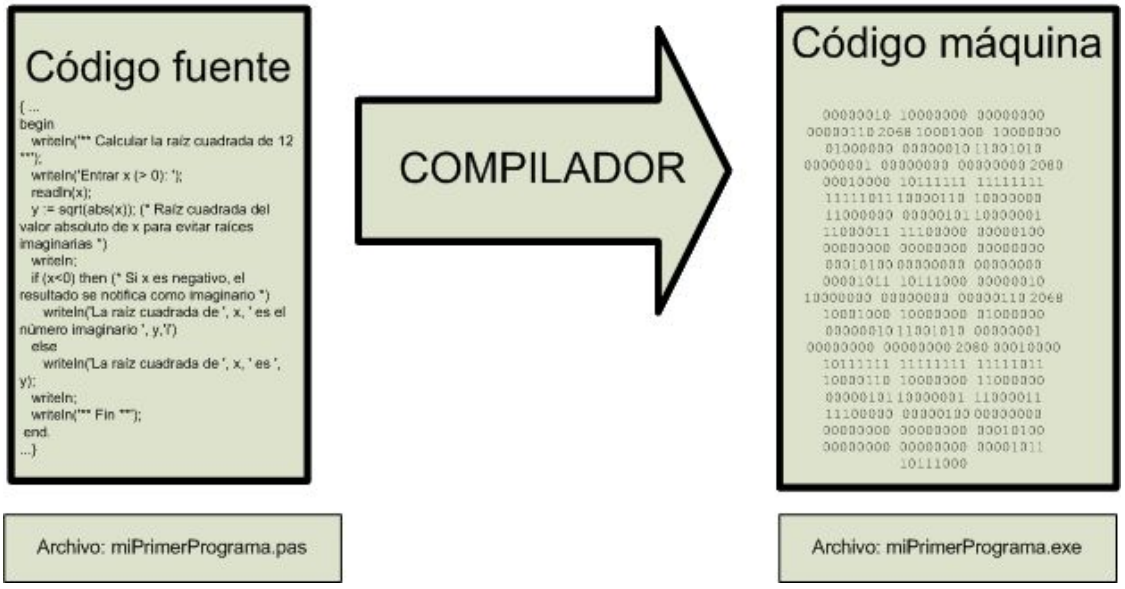
```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
        for  in n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ,'    %s -> {' % nodename
        for  in :namechildren
            print '%s' % name,
```

Son aquellos que, según sus reglas de escritura, se aproximan al lenguaje natural de las personas, utilizando habitualmente palabras clave y/o identificadores en inglés tales como: if, for, while, break, public, return etc. La mayoría de los programadores habitualmente trabajan con lenguajes de alto nivel. Dentro de los lenguajes de alto nivel tenemos todos los lenguajes

que utilizamos hoy día, en cualquier organización que desarrolle sistemas. A los lenguajes de alto nivel podemos sub-clasificarlos según el tipo de aplicación que está orientado a construir:

- Lenguajes para desarrollar aplicaciones de escritorio: C++, C#, Java, Ruby, Python y muchos más.
- Lenguajes para aplicaciones web: JavaScript, HTML, PHP, ASP, y muchos más.
- Lenguajes para “apps” Mobile: Java JME, Objective C, etc.

Lenguajes de programación “compilados” o "interpretados":



Antes que nada, definiremos el concepto de “ARCHIVO FUENTE”:

Los archivos fuente son aquellos archivos que tienen todas las instrucciones (código fuente) para poder luego, en base a éste, ejecutar de nuestro programa. Un archivo fuente puede tener 1 única instrucción, como también miles de ellas. Normalmente se suele escribir 1 instrucción por renglón (lo cual lleva a hablar muchas de veces de "cantidad de líneas de código" del archivo fuente).

Lenguajes compilados:

Son aquellos que generan (compilan) un nuevo archivo ejecutable (tal como un archivo .exe) a partir del archivo fuente. Una vez compilado nuestro programa, lo ejecutaremos a través del archivo resultante: .exe, .bat, u otros similares. Los programas una vez compilados, pasan a

estar escritos en un lenguaje de bajo nivel (habitualmente el lenguaje máquina). Ejemplos de lenguajes compilados: C#, Delphi, Cobol, Pascal, Fortran y muchos más.

IMPORTANTE: Si perdemos el archivo fuente NO será posible hacer futuros cambios en nuestro programa. NO PUEDE recuperarse el archivo fuente en base al archivo ejecutable ya compilado. Siempre debemos guardar muy bien nuestros archivos fuente.

Lenguajes interpretados:

Son aquellos que NO requieren compilación: Existe algún otro programa que es capaz de "interpretar" nuestro archivo fuente e ir ejecutándolo LINEA a LINEA. Los lenguajes interpretados son levemente más lentos que los compilados, ya que deben ir siendo traducidos a lenguaje máquina línea a línea, a la vez que el programa se va ejecutando. Resumiendo: La traducción a lenguaje máquina se hará una y otra vez, línea a línea, por cada ejecución del programa. Ejemplos de lenguajes interpretados: JavaScript, HTML, PHP, ASP, Perl y muchos más.

Lenguajes de programación case sensitive:

Los lenguajes de programación pueden ser “case sensitive” o NO serlos. Un lenguaje se considera “case sensitive” si es sensible a minúsculas o mayúsculas. Para estos lenguajes, algunas instrucciones deberán ser escritas en mayúsculas (o en minúsculas) para que funcionen, de lo contrario tendremos problemas de escritura que NO permitirán ejecutar correctamente el programa.

Herramientas del Programador

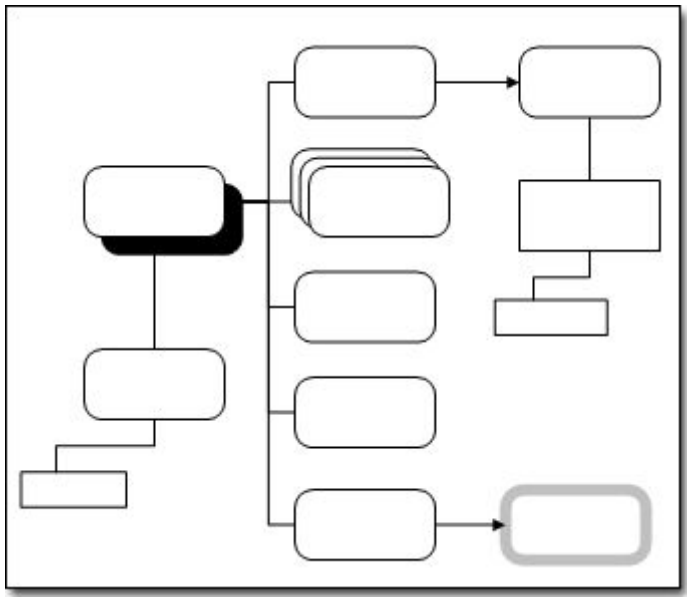
Pseudo-Código

El pseudo-código es un "código falso" que un procesador de computadora no puede ejecutar. Este código es similar al lenguaje natural de las personas, omite detalles técnicos que aparecen en el verdadero código; se trata de una aproximación al código que deberá ser escrito en un determinado lenguaje. El pseudo-código es un buen complemento a los diagramas de Jackson; ambos ideales en la etapa de aprendizaje en programación.

```
algoritmo Sumar

variables
  entero a, b, c

inicio
  escribir( "Introduzca el primer número (entero): " )
  leer( a )
  escribir( "Introduzca el segundo número (entero): " )
  leer( b )
  c ← a + b
  escribir( "La suma es: ", c )
fin
```

Diagramas de Jackson

Son esquemas / dibujos que ayudan a entender la lógica de nuestro programa. Estos diagramas se leen desde arriba hacia abajo (o bien en el orden que indiquen las flechas de secuencia); los cuales muestran el flujo de ejecución de la aplicación. Existen distintas entidades para distintos conceptos, tales como: rectángulos, rombos, cuadros, etc. Los diagramas de Jackson son sumamente útiles para aquellas personas que están empezando a estudiar programación.



Código

Por código entendemos al conjunto de instrucciones que ejecuta un procesador, los cuales deben ser escritas por los programadores. Un código luce similar a:

```
var nota1 = 7;

var nota1 = 5;

writeln('Primer nota: ' .nota1);

var promedio = (nota1 + nota2) / 2;

calcularNotas();
```

Pruebas de Escritorio

Son pequeños test, normalmente hechos en formato de tabla, y que se suelen escribir en papel para poder comprender que es lo que hace alguna pequeña funcionalidad de nuestro programa. En dicha tabla, se indicarán datos y valores de entrada, como también los datos de salida (es decir, los valores que uno espera obtener en base a los datos de entrada). Una funcionalidad de nuestro programa puede tener distintas combinaciones de entradas y salidas, de manera que nuestra prueba de escritorio estará formada por múltiples renglones.

ENTRADAS		SALIDA
Valor A	Valor B	SUMA
1	4	5
0	6	6
-1	3	2

Errores

Un error es todo aquello que genere y/o devuelva resultados INESPERADOS. Existen básicamente 3 tipos de errores:

Errores Sintácticos

Un error sintáctico es aquel que se comete cuando el código de nuestro programa está MAL ESCRITO: Errores de escritura (Olvidarse algún carácter de puntuación, escribir un carácter sobrante, etc). Podemos comparar un error sintáctico con un error “ortográfico”, propio del lenguaje natural de las personas. ¿Qué consecuencias trae un error sintáctico? En lenguajes compilados: Un error sintáctico NO permitirá que el programe se compile: Debemos corregir dichos errores para compilar el programa y luego poder ejecutarlo.

Se dice que los errores sintácticos ocurren en “tiempo de programación” o “tiempo de compilación”, lapso de tiempo en el cual se compila nuestro programa (a partir del archivo fuente) a un nuevo archivo ejecutable.

En lenguajes interpretados: Ocurren durante la ejecución del programa. Normalmente el programa se DETIENE al encontrar un error de escritura.

Los errores sintácticos son los errores más sencillos de resolver.

Errores en Ejecución

Un programa puede estar perfectamente bien escrito, libre de errores sintácticos, pero puede cometer errores durante su ejecución. Estos errores ocurren en “tiempo de ejecución” (“runtime”: Intervalo de tiempo que va desde que el programa inicia su ejecución hasta que finaliza). Este tipo de errores son producidos por acciones / operaciones imposibles de realizar (incompatibilidad entre tipos de dato y operadores u operaciones sin solución).

Ejemplos:

- Una división por cero.
- Cálculo de la raíz cuadrada de un número negativo.
- Bucles infinitos.
- Que el programa intente hacer un cálculo aritmético con valores de tipo string.
- Etc.

Los errores en tiempo de ejecución muchas veces DETIENEN nuestro programa: El programa ABORTA. A menudo usamos la expresión: “el programa se rompe”, “el programa pincha” cuando esto ocurre. Una causa frecuente a este tipo de problemas de debe a datos ingresados por el usuario; los cuales no han sido bien validados (Ejemplo: Se esperaba que el usuario ingrese un valor numérico, cuando en realidad ingresó una cadena de texto).

Errores Lógicos

Son aquellos relacionados con acciones inesperadas que comete nuestro programa. Para la computadora nuestro programa habrá sido ejecutado exitosamente; pero ante el usuario existirán problemas: El programa no actuó como se esperaba.

Ejemplos:

- Que el programa realice una suma aritmética cuando en realidad debió haber restado.
- Que el programa borre datos de la aplicación en lugar de agregar datos nuevos.
- Que el programa no arroje mensajes en pantalla cuando debió haberlo hecho.
- Etc.

Estos errores son los peores con los que nos podemos encontrar ya que requerirá una revisión en la lógica de alguna funcionalidad en particular o bien requerirá una revisión de toda la aplicación. Estos errores muchas veces están relacionados con en el pseudo-código o los diagramas de flujo que se armaron previamente al código real del programa. (La lógica de acción y ejecución del programa previamente pensada resulta que era EQUIVOCADA). Sea cual sea el tipo de error, siempre será deber del programador corregirlos.

El trabajo de buscar errores en nuestros programas se conoce como TESTING: Analizar y probar minuciosamente cada funcionalidad de nuestro programa tratando de descubrir errores que hasta ahora no habían salido a la luz.

Las pruebas de escritorio son una buena herramienta para hacer trabajo de testing; aunque también existen hoy día muchas herramientas automatizadas para hacer este tipo de trabajo: Tales como aplicaciones DEBUGGERS.

Debugger

Un debugger (depurador) es una aplicación complementaria al lenguaje de programación con el cual estemos trabajando, el cual nos ayuda a encontrar y solucionar posibles errores en el código de nuestro programa. Los debuggers nos permiten hacer pequeños de test de ejecución con el fin de detectar problemas que aún no han salido a la luz. Normalmente un debugger detiene su ejecución cuando encuentra algún error, el cual será informado marcando la sentencia errónea en color rojo o de alguna otra forma notoria.

Break Points

Los debuggers permiten hacer “break points”, los cuales son puntos de interrupción intermedios elegidos por el tester para examinar y analizar resultados parciales. Ejemplo: Hacerle un seguimiento al valor de una variable, el cual se presume erróneo. El debugger puede informarnos cuales son los valores iniciales de nuestras variables, y cuales son sus valores en el break point elegido. Muchos debuggers tienen botones del tipo: “play”, “pause” y “stop” los cuales permiten arrancar el test, pausarlo o directamente interrumpir el test de prueba. Hoy día existen muchos tipos de debuggers, cada uno enfocado a distintas tecnologías y lenguajes de programación. De NO usar debuggers encontrar errores puede ser un trabajo arduo y pesado, sobre todo en grandes aplicaciones con códigos extensos y complejos.

Interacción aplicación - usuario: Ventanas de Diálogo.

Muchas de las tecnologías actuales poseen lo que se conoce como “ventanas de diálogo”.

Estas ventanas permiten interactuar con el usuario de la aplicación en forma muy dinámica y elegante.

El usuario podrá a través de ellas:

- Ingresar datos con el teclado.
- Seleccionar opciones pre-establecidas con el mouse de la computadora.
- Confirmar o cancelar acciones.
- Recibir notificaciones de distinto tipo.

El uso de ventanas de diálogo genera aceptación, satisfacción y comodidad por parte del usuario final.

Conceptos de Programación

Comentarios

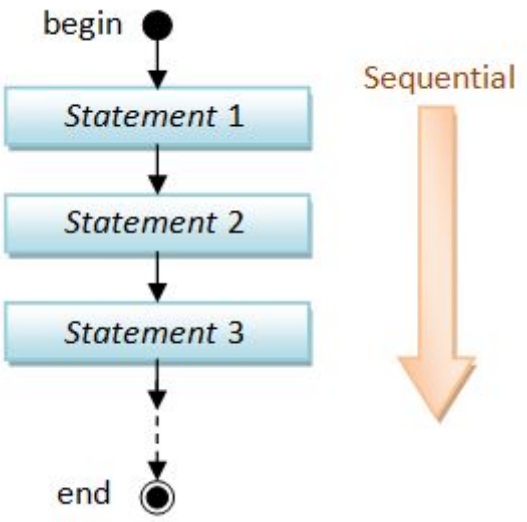
Los comentarios son anotaciones que escribe el programador en el código fuente del programa para explicar ciertas funcionalidades que pueden resultar difíciles de entender o recordar. Cualquier comentario es ignorado cuando el programa se ejecuta, por lo cual NO ocurrirán errores durante su ejecución. No existe límite a la cantidad de comentarios que pueden hacerse. Destacamos también que los comentarios son recursos útiles para anular de momentos la ejecución de código ya escrito, sin la necesidad de borrarlo. Tengamos en cuenta el siguiente ejemplo:

```
//archivo1.js
calcularIndice();

//archivo2.js
/* calcularIndice(); */
```

Como podemos apreciar, ambos archivos son bastante similares con la única diferencia en que en el archivo1.js la función calcularIndice esta libre en el programa mientras que en el archivo2.js la función se encuentra dentro de un comentario multilínea el cual evita que esta se ejecute en ese programa. Comentando un fragmento de código, se anula su ejecución por el motivo que fuere; cuando se desee volver a ejecutar dicho fragmento simplemente removeremos el comentario. Los símbolos para generar un comentario dependerán del lenguaje de programación con el que se trabaje.

Sentencias (Statements)

Una sentencia o statement es una ORDEN que le damos a nuestro programa, tales como: imprimir un dato en pantalla, guardar un dato en memoria, etc. Las sentencias componen las instrucciones (código) de nuestro programa.



En muchos lenguajes ciertos tipos de sentencias terminan con el signo de puntuación “;” en otros puede NO estar presente el “;”. A menudo veremos que se escribe 1 sentencia por cada renglón, dando un ENTER al final de la línea, tal como:

```
Sentencia1;
Sentencia2;
Sentencia3;
...
SentenciaN;
```

Aunque NO necesariamente tiene que ser así: Las sentencias podrían escribirse una al lado de la otra y nuestro programa funcionaría perfectamente: Y si bien funcionaría, en muchos casos este tipo de escritura puede generar dificultades a los programadores para comprender el código escrito:

```
Sentencia1; Sentencia2; Sentencia3; .... SentenciaN;
```

El código resultante es “muy horizontal” y difícil de seguir; sobre todo cuanto existen grandes cantidades de sentencias.

Orden de escritura de las sentencias

El ORDEN en el cual se van escribiendo las sentencias del programa NO es INDISTINTO. Las sentencias siempre se van ejecutando de ARRIBA hacia ABAJO. No resulta lo mismo:

Sentencia1;	(Ejemplo à imprimir: “Bienvenido al programa”)
Sentencia2;	(Ejemplo à imprimir: “Fin del programa”)

Que escribir:

Sentencia2;	(“Fin del programa”)
Sentencia1;	(“Bienvenido al programa”)

Las sentencias normalmente son dependientes de las acciones generadas por las sentencias anteriores, por lo cual escribirlas en un orden incorrecto generarán ERRORES GRAVES en la lógica de nuestro programa.

Variables

Una variable es una referencia a un espacio en memoria dentro de nuestra computadora. En dicho espacio en memoria, almacenaremos un dato. El concepto de variable debe verse como un contenedor de información. Toda variable tiene un nombre asociado, conocido como su

“identificador”. Ejemplos de identificadores: edad, apellido, cuil, etc. Por otra parte, tenemos los valores guardados en dichas variables:

21, “Perez”,”21-12345678-1”, etc.

Las variables son elementos fundamentales de la programación, ya que nos permitirá que nuestros programas guarden datos en memoria. El valor guardado en una variable siempre tiene un TIPO DE DATO asociado:

Podrá ser numérico, podrá ser un texto (cadenas de caracteres, “strings”) o bien tipos más específicos.

Tipos de Dato

Los tipos de dato nos indican con que clase de valor estamos trabajando. Claramente NO es lo mismo trabajar con valores numéricos que con valores de tipo texto. Cada tipo de dato tendrá determinadas operaciones permitidas. Por ejemplo: Los valores de tipo numéricos puedan sumarse, restarse mientras que un texto no lo podremos operar aritméticamente. Los tipos de dato pueden sub-clasificarse. Por ejemplo, los tipos numéricos los podemos dividir en tipos: enteros, positivos, negativos, decimales: valores en coma flotante, los cuales tienen parte entera y parte decimal. Los nombres exactos de los tipos de dato vienen dados por un lenguaje de programación específico. Por ejemplo, muchos lenguajes llaman “int” (integer) a los valores numéricos enteros que pertenezcan a cierto rango de valores (desde un valor X hasta un valor Y). Del tipo “int” se desprenden otros tipos más específicos: “tinyint”, “bigint”, etc los cuales son subconjuntos del tipo entero.

Recuerde: Habrá que estudiar un lenguaje de programación específico para conocer sus distintos tipos, pero a menudo encontramos los siguientes:

- “boolean” o bien “bool”: (Valores booleanos: true o false).
  Consumen 1 byte de memoria.

- “int” o bien “integer”: Valores enteros del rango 2.147.483.648 y 2.147.483.647.
  Consumen 4 bytes de memoria.

- “byte” o bien “tinyint”: Valores enteros del rango -128 a 127.
  Consumen 1 byte de memoria.

- “short” o bien “mediumint”: Valores enteros del rango -32.768 a 32.767

Consumen 2 bytes de memoria.

- “long” o bien “bigint”: Valores enteros del rango 9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.
  Consumen 8 bytes de memoria.

- “float” y “double”: Representan valores decimales, con distinto tipo de precisión en cuanto a la cantidad de cifras decimales.

Consumen 4 u 8 bytes de memoria.

- “string”: Cadenas de caracteres.
  La cantidad de memoria consumida dependerá de la longitud de la cadena.

- “char”: 1 único carácter.
  Consumen 1 byte de memoria.

Apartado Tipos de datos JavaScript

A continuación se listan los tipos de datos soportados en el lenguaje que veremos durante todo el curso:

- String : Vectores de caracteres o glifos ordenados secuencialmente entre comillas dobles o comillas simples. Ej.: “Educacion IT” , “1” , “Hola Mundo!”.
- Number : Cualquier expresión numerica ya que no distinguimos entre enteros o decimales en JavaScript. Ej.: 1, -1, 2.5.
- Boolean : Expresiones booleanas TRUE ó FALSE.
- Undefined : Toda variable declarada sin valor o cualquier propiedad interna no existente de un objeto
- NULL
- Object : Vector asociativo en n dimensiones. El mismo se inicializa de manera literal con {}(llaves) y contiene en su interior pares de indices asociados a valores por el operador : (dos puntos) separados por , (coma). Ej.: {nombre:”Educacion IT”}. Un objeto puede contener cualquier tipo de dato en su interior.
- Array : Objeto especializado en poder tener ademas de su comportamiento habitual la habilidad de guardar datos de manera secuencial, es decir bajo indices numericos auto incrementales. Los mismos se inicializan de manera literal con [](corchetes) y contienen en su interior cualquier tipo de dato se parado por , (coma). Ej.: [1,2,”Educacion IT”]
- Function : Es un objeto con la habilidad de poder ser invocado. Las expresiones funcionales que no retornen ningun valor tienen a valer undefined. Se inicializan usualmente con la palabra reservada function seguido por el nombre de la función, los

parentesis de argumentos y el cuerpo de la misma entre {}(llaves). Ej.: function foo(){console.log(“Hola Mundo!”);}. Se puede invocar una funcion escribiendo su nombre y pasando los parentesis de parametros pero es un tema que se tocará mas adelante en el curso.

¿Cómo definir una variable?

Dependerá del lenguaje que estemos utilizando. En algunos lenguajes habrá que especificar explícitamente qué tipo de dato guardan y en otros NO. Un ejemplo de escritura en Javascript es:

```
var edad;
```

Anteponiendo la palabra “var” estoy diciendo que “edad” es el identificador de una variable. Luego de definir la variable, debemos asignarle un valor. La asignación es la acción de guardar un valor en una variable; la cual debe hacerse a través del operador de asignación; el cual habitualmente suele ser el operador "=", pero dependerá del lenguaje que estemos usando. Ejemplo:

```
var edad = 30;
```

La asignación de un valor a una variable siempre se lee de DERECHA a IZQUIERDA.

edad	30

También pondremos al final de la línea el símbolo ; (punto y coma) el cual indica el final de la instrucción (sentencia). Anteriormente se explicaba: Algunos lenguajes no requieren el operador “;” para terminar una sentencia, pero es un símbolo recurrente en distintas tecnologías. A tener en cuenta:

El valor de toda variable podrá ser modificado a lo largo de la ejecución de nuestro programa (precisamente por eso que se llaman "variables"). En algunos lenguajes una variable NO puede cambiar el tipo de dato que guarda (números, textos, etc) y en otros lenguajes está perfectamente aceptado el cambio del tipo de valor en cuestión.

Variables y cálculos aritméticos

Si 2 variables guardan valores numéricos, entonces es posible realizar cálculos aritméticos entre ellas. Ejemplo:

```
var suma = var1 + var2;
```

Dentro de una variable auxiliar "suma" guardamos el valor de la suma entre 2 variables var1 y var2, las cuales deben haber sido previamente definidas con sus respectivos valores numéricos.

Operadores

Son símbolos (o un conjunto de símbolos) que permiten resolver determinadas operaciones. Un ejemplo son los operadores aritméticos, los cuales nos permiten resolver cálculos matemáticos. Algunos operadores se consideran BINARIOS, ya que requieren 2 operandos para poder hacer la operación (Ejemplo: La suma à 2 + 4). Otros operadores se consideran UNARIOS, ya que trabajan con 1 único operando. (Ejemplo: El operador de negación, el cual será visto en el siguiente módulo).

Operadores Aritméticos

+ (suma)

- (resta)

* (producto)

/ (división)

% MOD (resto / módulo)

El producto y la división (al igual que en matemática), tienen mayor precedencia que la suma y la resta. La precedencia es la prioridad natural con la cual los términos se van agrupando entre sí. Ejemplo:

2 + 4 * 2 devuelve: "10" y NO “12”

Distinto sería: (2 + 4) * 2 lo cual en ese caso sí devolverá: “12” (Con el uso de los paréntesis alteramos la precedencia por defecto).

Operador Módulo

Probablemente este operador nos resulte extraño, ya que no es utilizado en matemática. Este operador se utiliza para calcular el resto de la división entre 2 valores enteros. Tiene la misma precedencia que el producto y el cociente. Podemos decir entonces que:

11 RESTO 3 è 2

4 RESTO 2 è 0

La sintaxis del operador dependerá del lenguaje, aunque muchas veces nos encontramos con: "%" o bien "MOD":

11 % 3 è 2

11 MOD 3 è 0

Una gran utilidad de este operador es averiguar si un valor es PAR o IMPAR. En este caso, debemos dividir dicho valor POR 2. Al dividir por 2 un valor, los posibles restos son: 1 (Impar) ó 0 (Par).

Cadenas de Texto

Las cadenas de texto (también popularmente llamadas "strings") son un conjunto de caracteres que pueden conformar una palabra, una frase, etc. Ejemplos: "Buen dia", "Hola", "Tengo 50 años", " a + b = c". En programación las cadenas de texto se entrecomillan.

Concatenación de cadenas de texto

La operación asociada a las cadenas de texto es la concatenación. Concatenar es la acción de unir (juntar) cadenas, con el fin de generar una nueva. Por ejemplo:

"termo" + "metro" generará la cadena "termometro".

“metro” + “termo” generará la cadena “metrotermo”.

Como se puede apreciar, la concatenación NO es conmutativa. A + B NO resulta lo mismo que hacer B + A (Salvo casos excepcionales). La concatenación se lleva a cabo a través del operador correspondiente (operador de concatenación), el cual viene dado por un lenguaje de programación en particular. Podrá ser el operador “+”, el operador “.” u otros.

Expresiones

Se trata de un conjunto de valores (operandos) y operadores que generarán un resultado. Podemos tener expresiones tales como las aritméticas y las lógicas. Ejemplos:

2 + 2 (Expresión aritmética) è Resultado: 4

5 > 4 (Expresión lógica) è Resultado: true

NOTA: Las expresiones lógicas serán explicadas en detalle en el siguiente módulo.

Anexo JAVASCRIPT

JavaScript, a diferencia de las tecnologías web HTML y CSS, es un lenguaje de programación. Un lenguaje de programación provee distintas estructuras de control y funciones las cuales permiten dinamizar en gran manera los contenidos de nuestras páginas webs. No debemos confundir al lenguaje JavaScript como el lenguaje JAVA, los cuales son 2 lenguajes y tecnologías con propósitos totalmente diferentes.

Con Javascript podremos, entre otras cosas:

- Solicitar el ingreso de datos por parte del usuario.
- Arrojar ventanas emergentes con mensajes para el usuario.
- Obtener la fecha y hora actual.
- Crear animaciones y galerías de imágenes.
- Crear menúes desplegables.
- Ocultar elementos de la pantalla del navegador, como volver a mostrarlos nuevamente.
- Añadir dinámicamente nueva información dentro del cuerpo de la página web.
- Validar formularios.
- Crear juegos.

Y mucho más.

Características de JavaScript:

- Es una tecnología que corre en los distintos navegadores.
- Otorga dinamismo a una página web.
- Es un lenguaje case sensitive.
- Permite interactuar con ventanas de diálogo y eventos.
- Permite modificar la estructura de una página web, una vez que ésta es desplegada en la pantalla del navegador.

Uso e Inclusión de Javascript

Todo el código JavaScript de nuestra página HTML debe estar contenido dentro del tag de apertura:

<script type="text/javascript"> y el tag de cierre: </script>

<script type="text/javascript">

// Código JavaScript aquí

</script>

El cual puede ser ubicado dentro del <head> de la página o bien dentro del <body> dependiendo de nuestro caso en particular y de nuestra necesidad. Una variante consiste en incorporar nuestro código JavaScript dentro de un archivo externo .js el cual debe ser enlazado a una página HTML. Ejemplo:

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

Dentro del atributo src de la etiqueta script, indicaremos el path relativo del archivo .js con el cual queremos enlazar.

Comentarios

Al igual que en HTML, también contamos con la posibilidad de comentar nuestro código fuente. JavaScript provee 2 tipos posibles de comentarios:

Comentarios de línea:

// *un comentario*

Comentarios de bloque:

/*

contenido de

varias

lineas

***/**

Variables

Una variable se define anteponiéndole a su nombre asociado, la palabra "var". Ejemplo:

var edad ;

Por otra parte, al crear una variable, debemos asignarle un valor. La asignación debe hacerse a través del operador de asignación **=** (igual). Ejemplo:

var edad = 30;

Pondremos al final de la línea el símbolo **;** (punto y coma) por tratarse de una sentencia simple. Toda variable podrá ser modificada a lo largo de la ejecución de nuestro programa (precisamente por eso que se llaman "variables"). Si dos variables guardan valores numéricos, entonces es posible realizar cálculos aritméticos entre ellas. Ejemplo:

var suma = var1 + var2;

Dentro de una variable auxiliar "suma" guardamos el valor de la suma entre 2 variables var1 y var2 las cuales deben estar previamente definidas con sus respectivos valores numéricos. Si dos variables guardan valores string, entonces es posible concatenarlas. Ejemplo :

var a = “hola”;

var b = “Como estas?”;

var saludo = a + “ “ + b;

Operaciones básicas

Salida de datos en el cuerpo de la página:

```
/*document.write(valor)*/
document.write("Hola Mundo!");
```

Salida por consola:

```
console.log("Hola Mundo!");
```

Salida por ventana de alerta:

```
alert("Hola Mundo!");
```

Acceso al color de fondo de la pantalla:

```
document.style.backgroundColor = "red";
```