

MP0491

Sistemas de gestión empresarial
UF5 Desarrollo de componentes

**5.2. Lenguaje
proporcionado por los
sistemas ERP-CRM**

Índice

☰	Objetivos	3
☰	Características y sintaxis del lenguaje	4
☰	Tecnologías	9
☰	Declaración de datos y estructuras de programación	13
☰	Sentencias del lenguaje	18
☰	Resultado de nuestro programa	24
☰	Resumen	26

Objetivos

Con esta lección perseguimos los siguientes objetivos:

1

Conocer las principales normas de programación.

2

Conocer los lenguajes que se utilizan y saber cómo se utilizan.

3

Aprender a escribir un programa de forma clara y mantenible.

¡Ánimo y adelante!

Características y sintaxis del lenguaje

A lo largo del siguiente apartado vamos a mostrarte **cómo añadir componentes a tu ERP.**

Para ello, acudiremos a un lenguaje de programación que nos ofrecerá, o al menos establecerá, el paquete instalado y del que debemos respetar su sintaxis.

La sintaxis de una gran mayoría de los lenguajes de programación consiste en **oraciones de texto que incorporan números, palabras y signos de puntuación** (paréntesis (), corchetes [], llaves {}), etc.).

La sintaxis describe las **reglas y combinaciones entre los distintos elementos que forman parte del programa**, para definir un código que sea semántica y sintácticamente correcto. Consiste en un conjunto de símbolos interconectados entre si, junto con una colección de caracteres que siguen una determinada sintaxis establecida de antemano, que permite la transmisión de instrucciones al ordenador.

Esta redacción requiere una serie de normas, que cada lenguaje de programación define de manera predeterminada. Es decir: **cada lenguaje utiliza su propia sintaxis y su propio grupo de reglas.**

Existe un gran número de lenguajes, que podemos clasificar según diferentes criterios:

Por su nivel de abstracción

Es decir, por la relación entre las órdenes que se escriben, y la interpretación de esas órdenes a nivel máquina. Pueden ser:

- **De bajo nivel.** Son lenguajes totalmente dependientes de la máquina, de manera que un programa que se realiza con este tipo de lenguajes no se puede migrar o utilizar en otras máquinas.
- **De alto nivel.** Son aquellos que se encuentran más cercanos al lenguaje natural que al lenguaje máquina.

Por su paradigma

Es decir, por el estilo de desarrollo de programas. Este puede ser:

- **Imperativo.** Los programas se componen de un conjunto de sentencias que cambian su estado. Son secuencias de comandos que ordenan acciones a la computadora.
- **Declarativo.** Opuesto al imperativo. Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.
- **Lógico.** El problema se modela con enunciados de lógica de primer orden.
- **Funcional.** Los programas se componen de funciones, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.
- **Orientado a objetos.** El comportamiento del programa es llevado a cabo por objetos, que son entidades que representan elementos del problema a resolver, y tienen atributos y comportamiento.

Por su evolución

- **Primera generación:** lenguaje máquina.
- **Segunda generación:** se crean los primeros lenguajes ensambladores.
- **Tercera generación:** se crean los primeros lenguajes de alto nivel. Ej. C, Pascal, Cobol...
- **Cuarta generación:** se crean los lenguajes orientados a objetos, que hacen posible la reutilización de partes del código para otros programas. Ej. Visual, Natural Adabas,Java,PHP, etc.
- **Quinta generación:** se crean los lenguajes orientados a la inteligencia artificial. Estos lenguajes todavía están poco desarrollados. Ej. LISP .

Por su forma de ejecución

- **Lenguajes compilados.** Una vez escrito el programa, este se traduce a partir de su código fuente, por medio de un compilador, en un archivo ejecutable para una determinada plataforma. Para ello, se llevan a cabo una serie de pasos:
 - **Análisis del programa fuente.** El compilador comprueba que se ha escrito correctamente.
 - Análisis de léxico.
 - Análisis sintáctico.
 - Análisis semántico.
 - **Síntesis del programa objeto.** Tiene como meta la generación de un código que pueda ser ejecutado en la máquina destino. Para mejorar la adaptación, puede disponer de fases de *linkeditacion*, donde añadiríamos las librerías necesarias.
- **Lenguajes interpretados.** Requieren un intérprete para implementar o ejecutar el código . Para mejorar el rendimiento, hay intérpretes que realizan una pseudo-compilación y guardan el resultado, detectando cambios en la fuente para repetir la compilación. Es un tema que no nos debe preocupar porque, en estos casos, trabajamos con las fuentes, y es el sistema quien se encarga de tomar las decisiones adecuadas.

Elegir un lenguaje

Cuando tenemos que escribir un nuevo componente para un ERP, no solemos disponer de demasiadas posibilidades de seleccionar el lenguaje.

(i) Recuerda que el lenguaje que utilizemos no tiene por qué ser el mismo que se utilizó para desarrollar el ERP, sino el que los creadores del ERP planificaron como extensión (aunque en la mayoría de casos coinciden).

Veamos los lenguajes que utilizan algunas de las distribuciones de ERP/CRM:

Dolibarr

Todo el paquete está desarrollado en PHP, y también es el lenguaje que utiliza para programar sus componentes. Dado que utiliza PHP sin ningún tipo de *framework*, quizás sea uno de los más sencillos de interactuar (siempre que sepas programar PHP...). Podemos utilizar HTML y javascript a nuestro gusto.

SAP

Esta empresa creó su propio lenguaje, tanto para programar la aplicación, como para programar los componentes ABAP (*Advanced Business Application Programming*). Es un lenguaje de programación de cuarta generación, utilizado para fines de desarrollo y personalización en el software SAP.

PeopleSoft

Este ERP, gestionado en la actualidad por Oracle, tiene su propio lenguaje de programación (PeopleCode). Se trata de un lenguaje orientado a objetos, con una importante librería de funciones que resulta muy útil para facilitar el trabajo.

Siebel

Se trata de otro CRM distribuido desde 2006 por Oracle, y que se puede programar con eScript (muy semejante a Javascript) y con Siebel VB (semejante sintáctica y semánticamente a Visual Basic de Microsoft). Ten en cuenta que SiebelVB solo corre en máquinas Windows.

JD Edwards

Un ERP más comprado por Oracle en 2006 y distribuido por ellos. En la versión antigua, para AS400, utilizaba exclusivamente RPG; en la última versión, resuelven casi todo a través de medios gráficos, que permiten montar componentes arrastrando y clicando, y, si se tiene que escribir algo de código (solo para reglas de negocio), se puede utilizar NER (*Named Event Rule*) o C (*Business Function*).

OpenBravo

Un ERP desarrollado sobre Java y que acepta componentes en Java. Buena solución en el actual estado de este lenguaje.

Odoo

ERP y CRM desarrollado en Python y que utiliza ese mismo lenguaje para los componentes. Tiene su propio marco API y su propio lenguaje de plantillas: QWeb. Es necesario conocer XML. Las habilidades de HTML y diseño web también son muy útiles, pero el 90% del trabajo se refiere a Python y XML.

Microsoft Dynamics AX

ERP de Microsoft, comprado a Navision. El desarrollo y modificación del software se realiza mediante su propio entorno de desarrollo integrado, **MorphX**. El entorno de desarrollo permanece en la misma aplicación del cliente, permitiendo de esta forma tener acceso a dichas herramientas desde la aplicación del cliente. El lenguaje que se emplea (X++) es muy similar a Java o a C++.

Como ves, podríamos crear dos grupos:

1

Paquetes que utilizan un lenguaje propio. Como desventaja tienen que no habrá muchos programadores que lo conozcan y lo más probable es que tengamos que formarlo nosotros. Como ventaja, que son lenguajes totalmente orientados a lo que tienen que hacer, por lo que trabajar con ellos será más fácil.

2

Paquetes que utilizan un lenguaje de programación existente. En este caso, ocurre exactamente lo contrario. En función del lenguaje, nos podemos encontrar con muchos programadores que solo requerirán una pequeña adaptación para conocer las API y el entorno; sin embargo, al trabajar con un lenguaje estándar, puede que cueste más realizar las acciones que quieras.

Tecnologías

Un apartado que debemos tener en cuenta cuando vamos a crear nuestro componente es **bajo qué estructura estamos trabajando.**

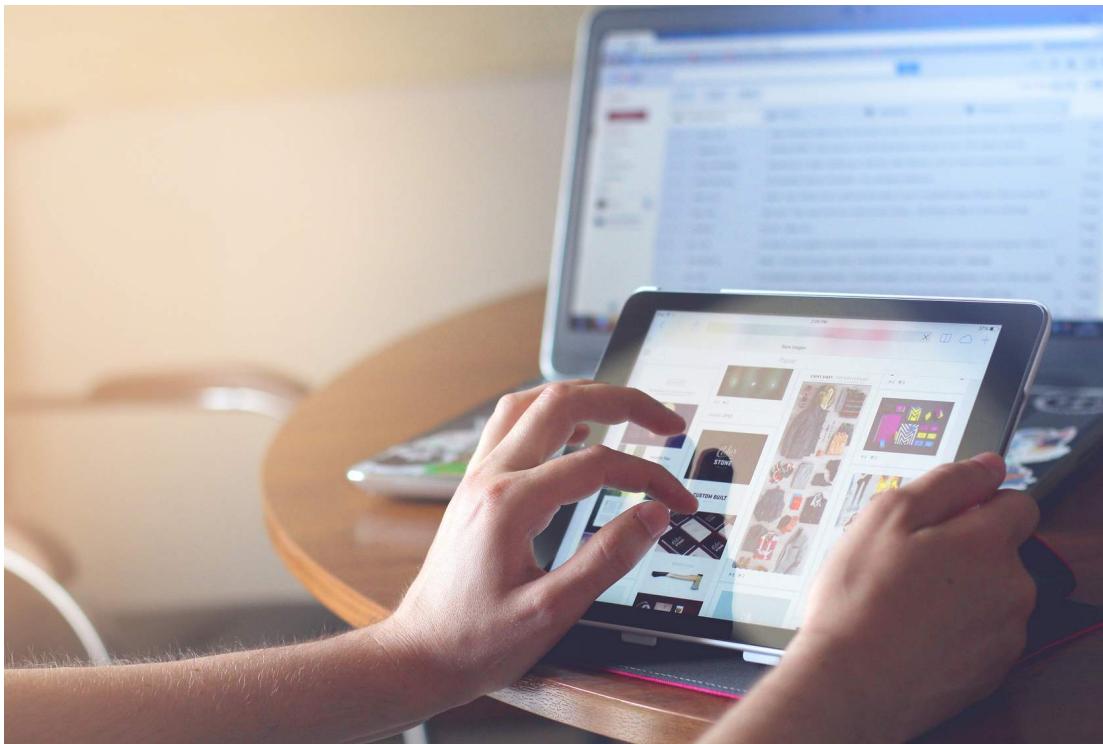
En la mayoría de casos, la aplicación ERP/CRM tiene que trabajar de forma que **se puedan conectar muchos clientes y compartir la información.**

Ese requerimiento solo obliga a que la base de datos sea única. No importa cómo esté resuelto, ya sea porque solo existe en un equipo servidor, o porque hay montada toda una infraestructura. **Aunque la BBDD reside en varios servidores, los programas la ven como única.**

Para trabajar con una base de datos centralizada podemos instalar los programas en cada ordenador y que sean ellos los que busquen la información de la BBDD. Pero esta tecnología tiene algunos **problemas:**

- Todos los ordenadores han de tener **potencia para gestionar el ERP/CRM.**
- Todos los ordenadores han de tener **el mismo S.O., y puede que la misma versión,** ya que, de no ser así, deberemos mantener el programa compilado para varias plataformas.

- La compartición de datos solo la puede soportar la base de datos, por lo que perderemos eficiencia.
- Los programas no se comunican entre ellos, por lo que cualquier comunicación tendremos que realizarla a través de la base de datos.
- Cuando haya una revisión de versión, deberemos actualizar todos los programas (de todos los puestos). Si esa modificación conlleva cambios en la base de datos, tendremos que detener todo el procesamiento, a nivel de empresa, para realizar la actualización de los ordenadores y la base de datos antes de poder autorizar que continúe la gestión.



Por otro lado, podemos dividir los programas entre **los que dialogan con el usuario (pantallas) y los que procesan datos**.

Eso hace que nuestras máquinas remotas sean más ligeras y que se suavice el problema de mantenimiento; sin embargo, nos obliga a introducir las comunicaciones dentro del desarrollo y también a que nuestras máquinas tengan el mismo S.O.

Desarrollo de los navegadores

En estos momentos, asistimos a un desarrollo de estas herramientas que las hace ideales.:

- Hay navegadores para todos los sistemas operativos y para todas las máquinas.
- El entorno de programación dentro de ellos es estándar.
- Su mantenimiento no afecta a las funcionalidades del paquete, ni tampoco a la BBDD.
- Funcionan en múltiples dispositivos, desde en un teléfono móvil, hasta el mayor ordenador personal.

El usuario actúa sobre pantallas dibujadas por navegadores en los que toda la programación reside en el servidor, con las consiguientes ventajas. La única dificultad es una **filosofía cliente-servidor llevada al extremo, en la que necesitamos continuamente la comunicación y el diálogo**.

i Con los nuevos navegadores ya no tenemos que controlar las comunicaciones porque lo hace la propia herramienta, pero tenemos que pensar siempre en ello.

Ejemplo

Te plantearemos un ejemplo sencillo.

Imagina que tenemos una pantalla que solicita al usuario seleccionar el país, la provincia y la población. Si queremos ayudarle, podemos presentarle la lista de países, una solución genial. Y, cuando el cliente seleccione el país, deberíamos presentarle las provincias. Pero, para eso, necesitaremos acudir a la BBDD con el país y pedir una lista de las provincias.

Hacerlo en un entorno con nuestro programa, no es ningún problema. Sin embargo, **en un entorno cliente-servidor debemos enviar la petición al servidor, y el servidor responderá con la lista.** Para nosotros, significará una pantalla nueva o, incluso, utilizar AJAX para actualizar la pantalla actual.

Es más complicado, pero totalmente universal, ya que gracias a esta estructura los usuarios pueden trabajar desde el móvil, un ciber café, su casa o la oficina sin que tengamos que modificar nuestro programa.

Declaración de datos y estructuras de programación

A lo largo de este apartado vamos a intentar contestar a una pregunta: **¿Cómo se organizan los datos?**

Cada solución ERP contiene una serie de herramientas que permiten el acceso a los datos, independientemente de su procedencia o formato.

Se trata de herramientas que garantizan la seguridad en sus movimientos. De esta manera, todos los datos se consultan de forma segura, con los permisos que tenga el usuario.

Declaración de datos

Nunca vamos a acceder a una tabla directamente, sino que disponemos de herramientas para pedirle al sistema que realice la consulta y que nos entregue los datos. De esa forma, conseguimos que nuestro acceso quede dentro de las normas de seguridad detalladas para la aplicación.

- i** Recuerda que nuestro ERP/CRM contiene todos los datos de la empresa una información que nos hace mejores y nos ayuda en la toma de decisiones futuras. ¡Imagina lo importante que puede ser toda esa información si estuviera accesible para la competencia!

Tipos de campo en un entorno ERP

Cuando estamos realizando una consulta, el tipo de datos de cada campo estará definido por el programador que creó la tabla, o por el que programó la consulta. Pero puede interesarnos cambiar el tipo de campo, crear campos de cara al listado o a la salida, o cualquier otra modificación. A continuación te mostraremos los **tipos de campo presentes en cualquier entorno ERP**.

Los tipos de campo especifican el contenido que tiene el campo, es decir, si se trata de un campo numérico, alfabético... El tipo disponible depende de cada paquete y de cada base de datos, aunque sus funcionalidades vienen a ser las mismas. Veamos los **distintos tipos de campo y lo que pueden contener**:

- **boolean**: booleano; solo puede contener los valores 1 o 0, que también son los valores lógicos para *verdadero* y *falso*.
- **integer**: entero, es decir, que puede contener un valor numérico entero (sin decimales).
- **float**: de coma flotante, que puede contener un numero con decimales. Puede que tengamos que indicar el número de posiciones decimales que queremos que contenga.
- **char**: carácter; puede contener cadenas de caracteres de la longitud que especifiquemos. Debemos indicar la longitud que deseamos que tenga.
- **text**: texto, un campo de texto de longitud ilimitada.
- **date**: fecha, que contiene una fecha.
- **datetime**: fecha-hora.
- **binary**: binario, que contiene una cadena de datos binarios. No tiene un criterio de selección especificado.
- **Array o vector**: lista de campos del mismo tipo a los que se accederá por medio de un índice. Normalmente, un vector tiene un número finito de elementos que se establece cuando se crea.
- **Lista**: se trata de un vector donde se puede modificar dinámicamente el número de elementos.
 - Tanto los *arrays* como los *vectores* pueden estar indexados en base 0, cuando el primer elemento del *array* es el que esta en la posición 0, o en base 1, cuando el primer elemento del *array* esta en la posición 1.

Aparte de esta lista de tipos que especifica el contenido del campo, podemos encontrarnos con algún tipo más, que se define en función del uso, como por ejemplo:

- **table.** Puede contener todo el contenido de una consulta.
- **row.** Contiene el valor de todos los campos de una fila seleccionados en la consulta.
- **cursor.** Cuando recibimos el resultado de una consulta fila a fila, este campo apunta a la fila activa y nos permite movernos por toda la consulta, haciéndolo avanzar o retroceder.
- **onezone.** Indica que este campo es una clave foránea y que va a otra tabla con la que mantiene una relación 1X1.
- **one2many:** Indica que este campo es una clave foránea y que va a otra tabla con la que mantiene una relación 1X varios.
- **many2many:** Indica que este campo es una clave foránea y que va a otra tabla con la que mantiene una relación varios a varios



Lenguaje y tipo

Hemos descrito los distintos tipos de datos y aprendido que debemos ser exactos con ellos. Pero hay lenguajes que no tienen en cuenta el tipo de datos (de tipado débil). Por ejemplo, si tenemos que programar en javascript, podemos definir un campo y ni siquiera asignar el tipo de dato. Ese campo tendrá el tipo en función del valor que contenga: cuando contenga números, sera *integer* o *float*, y cuando contenga caracteres, sera *char*. Y lo mismo pasa con PHP o con Python.

En cambio, con Java (de tipado fuerte), al intentar poner letras en un campo numérico nos dará un mensaje de error. Además, todo esto nos pasa en el componente, en las instrucciones que escribamos, pero **no** en la base de datos.



Cuando se define una columna en una base de datos, se especifica el tipo de datos que va a contener; y cuando se va a realizar una inserción o una actualización, el tipo de datos se comprueba, indicando *error* si no coincide.

Por eso nosotros, al programar un componente, **definiremos el tipo de dato del campo y nos mantendremos fieles al mismo**, aunque el lenguaje permita otras cosas.

Estructuras

En función de los campos que podemos crear, las estructuras pueden ser de dos tipos:

- **simples**: campos cuyo contenido se puede almacenar directamente en la tabla, como son los booleanos, los numéricos...
- **relacionales**: campos que nos permiten establecer una relación entre este registro y el registro de otra tabla, como los *many2one*, *many2many*, *one2one*...

Si juntamos varios campos en una lista crearemos un "Modelo"; llamamos "Modelo" a una colección de campos que representan en memoria la estructura de una tabla. Todos estos campos deben estar relacionados, como, por ejemplo, todos los datos de un cliente.

En todo modelo y para todas las tablas, el motor ERP/CRM acostumbra a crear **cinco campos mas**:

1

id (id): identificador único del registro dentro de la tabla.

2

create_date (DateTime): fecha y hora en que se creó el registro.

3

create_uid (Manyzone): identificador del usuario que creó el registro.

4

write_date (DateTime): fecha y hora de la última modificación del registro.

5

write_uid: identificador del último usuario que modificó el registro.

A continuación te mostraremos las sentencias del lenguaje.

Sentencia del lenguaje

Y, por fin, ha llegado el momento de describir el programa.

Ya sabemos qué son los programas y las estructuras de datos, pero, ¿cómo vamos a utilizar todo eso para hacer nuestros nuevos componentes? Recuerda que según el concepto de **herencia** no modificamos los componentes originales, solo heredamos de ellos.

Cuando hablamos de **crear un componente**, nos referimos a cualquier cosa que altere (normalmente añadiendo) funcionalidades a nuestro paquete ERP/CRM. Y consideramos "componente" también cuando añadimos un formulario o un informe. Sin embargo, a partir de este momento consideraremos que un componente está formado por uno o varios programas, y nos referiremos únicamente al programa.

Mantener un programa

Si tienes experiencia en programación, ya sabes que lo difícil de un programa no es hacerlo, si no mantenerlo. Un programa lo escribimos en un tiempo relativamente corto, sobre todo si lo comparamos con el tiempo que previsiblemente se utilizará ese programa.

Durante ese intervalo de tiempo, que puede durar años, en el que nuestro programa estará funcionando, pueden surgir mil ocasiones para modificarlo, bien porque se nos haya escapado algún error, bien porque hayan aparecido nuevos condicionamientos o leyes que impliquen modificaciones.

Cuando llegue el momento de abrir el programa fuente para localizar el punto problemático y realizar la modificación, **puede que nosotros ya no estemos trabajando para esa empresa**. Y aunque fuera así, es muy difícil que pasado uno o dos años consigamos recordar claramente todos los programas que hicimos y por qué, exactamente, escribimos determinada linea.

Este es otro de los motivos para **escribir los programas con la máxima claridad posible**. Y para lograrlo, hemos de mantener en la cabeza algunas normas.

Programas y funciones

- **Hemos de hacer que los programas puedan ordenarse** por lo que hacen. Es decir, los programas han de estar relacionados con un objeto y dedicarse solo a ese objeto.
 - Podemos escribir un programa que se dedique a listar clientes de varias maneras, y otro distinto que se dedique a listar artículos de varias maneras, pero no uno que lo haga todo.
- **Podemos escribir funciones dentro de los programas.** Una función es una parte del programa dedicada a realizar una sola acción.
 - Podemos escribir una función para calcular la letra del DNI, otra para determinar la edad del cliente, y así sucesivamente.
- **Han de ser pequeños.** Tanto los programas como las funciones no deben aspirar a "resolverlo todo"; no dudes en llamar a **subfunciones**, no solo porque el código se pueda repetir, sino, incluso, por la claridad. Antes hemos hablado de una función para calcular la letra del DNI; puede que solo se llame una vez a esta función, pero es mucho más claro apartar todo su código a un solo punto, que encontrárselo en medio.

Instrucciones

- **Indentación del código.** Permite la visualización de las instrucciones que están condicionadas a otras, permitiendo una lectura mucho mas cómoda del programa. Observa el ejemplo que nos propone [wikibooks](#):

Ejemplo de código sin Indentar

```
1 coger plato
2 echar jabon
3 pasar el estropajo por el plato
4 si hay suciedad ir a la instrucción 2
5 si el plato no es azul ir a la instrucción 7
6 ponerlo con los azules
7 si hay más platos ir a la instrucción 1
```

Ejemplo de código Indentado

```
mientras haya platos
```

```
    coger plato
    mientras haya suciedad
        echar jabon
        pasar el estropajo por el plato
    si plato es azulado
```

```
    ponerlo con los azules
```

- **En la programación estructurada hay un inicio y un fin perfectamente bien definidos.** Un solo inicio y un solo fin, ya que debemos evitar salir de un programa o de una función antes de que finalice.
- **Solo habrá una instrucción en cada línea.** Escribiremos el programa para conseguir la máxima claridad.
- **Utilizaremos paréntesis** para cerrar operaciones matemáticas y controlar así el orden de ejecución.
- **Separaremos los operadores binarios con espacios a ambos lados,** ya que de este modo se resalta el operador y se facilita la lectura del programa.

Variables

- **Los nombres de los campos han de explicar lo que contienen.** Puedo llamar a un campo "b", y ponerle dentro el precio del producto para luego operar. Pero si hago eso cada vez que veo "b", tendré que averiguar qué contiene el campo.
- **Reservaremos las variables de una letra** para su utilización en índices y contadores.
- **Los nombres deben estar en minúsculas**, excepto la primera letra de cada palabra a partir de la segunda (estructura *lower camelCase*). La alternativa, perfectamente válida, es separar las palabras del nombre con un guión inferior; ambos sistemas te permiten leer el nombre de la variable (pero el segundo método puede ser mas útil en entornos de BBDD).

precio_venta o *precioVenta* pueden ser ejemplos válidos.

- **Las constantes** se deben indicar al principio del programa y asociarlas a una variable con todas las letras en mayúsculas.

Por ejemplo: CAMBIO_EURO = 166.386

- **Evitar el uso de variables globales.** Si nuestro lenguaje soporta variables globales, solo deberíamos utilizar aquellas definidas por el propio lenguaje, ya que las que nosotros definimos son frágiles. Es necesario que el programa que las define haya pasado, que nadie las haya modificado, etc.

Comentarios

- **Todos los programas y funciones deben tener un comentario que explique qué hacen.** Este comentario puede utilizarse para documentar la aplicación, ya que explica lo que entra, lo que sale y lo que hace. Aparte, y de forma interna, podemos tener un segundo comentario que explique cómo lo hace, si es complejo. Este comentario es para los que tengan que trabajar con el programa, no para los que lo utilizan.
- **Al inicio del programa,** se debería indicar en el comentario, además de su descripción, la fecha de creación, y, para cada modificación, la fecha de modificación, la persona que la realizó y el motivo para realizarla.
- **Separar con una línea** el código del comentario.
- **No debemos comentar obviedades,** pero sí todo aquello que podemos olvidar fácilmente o que no es totalmente transparente.

HTML

Mientras que el lenguaje a utilizar dependerá de la plataforma elegida, tienes muchas posibilidades de escribir código en HTML. Estas son algunas "buenas costumbres" para ese código:

- Los *tags* tienen que estar escritos en minúsculas. Por ejemplo, `<td>` en vez de `<TD>`.
- Los objetos html deben tener *id* y *name*, y ambos deben ser iguales. Cuando solo se pone el *name* y en IE se usa `getElementById` funciona (incorrectamente), pero no en FireFox. Para evitarlo se deben usar ambos, como en el siguiente ejemplo:

```
<input id="nombre" pre="" name="nombre" type="text" />
```

- Cuando hay que escribir variables de lenguaje dentro de un código HTML, se debe tener el HTML dentro de lenguaje, y no al revés.
- Finalmente, el código HTML generado por un programa también debe ser legible, bien estructurado e indentado. Para ello, se recomienda el uso de "/n" y "/t" .

Codificar estructuras de control

1

Estructuras de control condicionales

Se trata de estructuras de control que permiten **redirigir un curso de acción según la evaluación de una condición simple, ya sea falsa o verdadera**.

Si la condición es verdadera, se ejecuta el bloque de sentencias 1; de lo contrario, se ejecuta el bloque de sentencias 2:

```
IF (Condición) THEN
    (Bloque de sentencias 1)
ELSE
    (Bloque de sentencias 2)
END IF
```

Se pueden plantear múltiples condiciones simultáneamente. Por ejemplo: si se cumple la (Condición 1) se ejecuta (Bloque de sentencias 1). En caso contrario, se comprueba la (Condición 2); si es cierta, se ejecuta (Bloque de sentencias 2), y así sucesivamente hasta n condiciones. Si ninguna de ellas se cumple, se ejecuta (Bloque de sentencias *else*).

```
IF (Condición 1) THEN
    (Bloque de sentencias 1)
ELSEIF (Condición 2) THEN
    (Bloque de sentencias 2)

    .....

ELSEIF (Condición n) THEN
    (Bloque de sentencias n)
ELSE
    (Bloque de sentencias else)
END IF
```

2

Estructuras de control en bucles

Permiten que se ejecuten repetidamente un conjunto de instrucciones, un numero predefinido de veces, o bien hasta que se satisfaga una determinada condición.

Satisfacer una determinada condición consiste en que el resultado de una expresión lógica sea "true", Los métodos más comunes son las comparaciones entre dos variables ($a = b$) o ($a > b$)...

Resultado de nuestro programa

Y, por fin, en este apartado escribiremos el programa.

Hemos conseguido escribir el programa (si no ahora, lo podrás hacer en cuanto tengas el ERP definido y surja la necesidad). Una vez escrito, debemos revisarlo para ver que cumple con todos los requisitos.

Centraremos nuestra revisión en cuatro puntos fundamentales:

1

Corrección

Un programa se considera correcto cuando **cumple con todos los requerimientos y funciona tal y como se concibió en sus orígenes**. Esto hace totalmente imprescindible que, antes de escribir ni una sola línea, tengamos claro lo que tenemos que hacer y cómo lo tenemos que hacer, y que lo dejemos plasmado en un documento, que podremos consultar.

2

Claridad

Es fundamental que el programa sea **lo más sencillo e interpretable (entendible) posible**. Es necesario que su estructura sea tan legible y coherente como nos sea posible. Su estilo de edición debe ser cuidado al milímetro, ya que es indispensable facilitar el trabajo del programador, de cara a las fases posteriores de corrección de errores, modificaciones, ampliaciones, etc.

i Recuerda que no siempre un programa es modificado por el mismo programador. Es posible que el que codifique las pautas o corrija errores sea otro programador distinto.

3

Eficiencia

No solo se pretende que el programa sea capaz de funcionar en las condiciones pactadas de antemano. Se trata también de que gestione, de la mejor forma posible, los recursos que utiliza. Cuando se habla de eficiencia, se hace mención al tiempo que tarda en ejecutarse y a la cantidad de memoria que precisa, aunque también hay que tener en cuenta el espacio que necesita en disco u otros recursos como tráfico de red, memoria, etc.

4

Portabilidad

La portabilidad es una propiedad muy valorada actualmente entre los programadores, teniendo en cuenta que cada vez existen mas plataformas GNU/LINUX, de distribución libre, que están irrumpiendo en el mercado con gran éxito. Lo deseable es que nuestros programas sean también ejecutables en la familia de SO Windows.

Lenguajes como Java, que puede correr prácticamente en cualquier plataforma, hacen mucho más interesante el desarrollo, aunque solo afecte a la parte del servidor.

Resumen

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- A lo largo de esta unidad hemos conocido los **criterios de clasificación de los lenguajes** por su nivel de abstracción, por su paradigma, por su evolución o por su forma de ejecución.
- Sabemos que el lenguaje que podemos utilizar dependerá de la elección de nuestro **ERP/CRM**.
- Hemos aprendido algo más acerca de las tecnologías cliente-servidor, así como del condicionamiento que representa trabajar con ellas.
- También hemos analizado los **tipos de datos**, simples y complejos, y las **estructuras de datos** que nos podemos encontrar.
- Y, por último, hemos revisado algunas **normas de programación** que nos ayudarán a escribir nuestros programas, y a permitir que personas ajena a nosotros puedan mantenerlos.



PROEDUCA