

Liceo Compu-Market

Catedrático: Erick Gonzales

Catedra: Programación

Tema:

Programación

Nombre: Luisa Fernanda

Amezquita Castro

Grado: 5to Bach. “A”

**Índice**

**No. Descripción No. De Pág.**

**1 Carátula 1**

**2 Índice 2-3**

**3 Introducción 4**

**4 Programación 5-6**

**4.1 BASIC o Ruby.**

**5 Fases para la creación de un programa 7-8**

**5.1 Definición del Problema**

**5.2 Análisis del Problema**

**5.3 Diseño de algoritmos**

**5.4 Codificación**

**5.5 Prueba de depuración**

**5.6 Documentación**

**5.7 Mantenimiento**

**6 Programación didáctica 9**

**7 Léxico y programación 10**

**8 Programas y algoritmo 11**

**9 Código (programación / código abierto y cerrado) 12**

**10 Código fuente 12**

**11 Lenguajes de programación 12-13**

**11.1 Basic.**

**11.2 C.**

**11.3 C++.**

**11.4 PHP.**

**12 Lenguaje de programación 14**

**12.1 Tipos de lenguaje de programación 14**

**13 Entorno de desarrollo integrado 15**

**14 Compilación: hacerlo comprensible a la computadora 15-16**

**15 Sintaxis 16**

**16 JavaScript 17**

**17 Java 18-21**

**18. El lenguaje de programación Java**

**18.1 Como trabaja java**

**19 Angular JS 22**

**20 Comandos básicos de SQL 23**

**21 Booststrap 24**

**22 Eventos JQuery 25**

**23 Método CRUD 26**

**24 Método BOF 27**

**25 SCRUM 28-30**

**25.1 SCRUM MASTER 30-31**

**26 Metodología de programación 31-32**

**26.1 Programación estructurada**

**27 Definición de las 3 estructuras básicas 32-33**

**27.1 Estructura 27.2 Estructura selectiva**

**28 Ventajas de la programación estructurada 34-35**

**29 Programación orientada a objetos 36-39**

**30 Tipos de datos primitivos y operadores 40-49**

**31 Estructura de control: IF, FOR, FOR EACH, WHILE 50-51**

**32 Programación modular 52-53**

**33 Técnicas básicas de programación 53-54**

Programación

Programación es el proceso a través del cual un programa o aplicación informática es desarrollado.  
Se conoce como programación en ciencias de la computación a los pasos que se abordan para crear el código fuente de un programa informático. De acuerdo con estos pasos, el código se escribe, se prueba y se perfecciona.  
  
El software informático es aquel ejecutado por el hardware o dispositivos de una computadora, permitiendo que el usuario acceda a funciones y operaciones de todo tipo. Así, la programación es una de las actividades más determinantes en el desarrollo de sistemas eficientes, rápidos y amigables para todo tipo de usuarios.

Se conoce como programadores a aquello encargados de desarrollar este código con instrucciones para que el software se comporte de una u otra manera de acuerdo con las órdenes que reciba. El lenguaje de programación es, por otro lado, la serie de parámetros y códigos de los que el programador se vale para desarrollar software. Existen distintos tipos de lenguajes, como el C,   
  
BASIC o Ruby.  
  
Además, existe la ingeniería el software, que se dedica a desarrollar modelos de software para programas de gran envergadura.  
  
Típicamente, para programar un software o aplicación, el programador debe en principio reconocer el principal problema o tarea a la que se destinará el programa, definir los requisitos y tipo de funcionamiento, diseñar la arquitectura, implementar el programa, implantarlo o instalarlo y, luego, perfeccionarlo sobre la base de pruebas y errores.  
  
Hoy en día existen todo tipo de lenguajes de programación, algunos más sencillos o que tienen el propósito de facilitar la tarea de desarrollar pequeñas aplicaciones. Entre ellos, Ruby es uno de los más popularizados en los últimos años, desarrollado por un programador japonés y que combina sintaxis de distintos lenguajes como Python o Perl.  
  
Así, prácticamente cualquier usuario informático puede adquirir algunas nociones de programación y desarrollar aplicaciones a su medida.

Lenguaje de [programación](http://www.monografias.com/Computacion/Programacion/): [Sistema](http://www.monografias.com/trabajos11/teosis/teosis.shtml) de [símbolos](http://www.monografias.com/trabajos36/signos-simbolos/signos-simbolos.shtml) y reglas que permite la [construcción](http://www.monografias.com/trabajos35/materiales-construccion/materiales-construccion.shtml) de programas con los que [la computadora](http://www.monografias.com/trabajos15/computadoras/computadoras.shtml) puede operar así como resolver [problemas](http://www.monografias.com/trabajos15/calidad-serv/calidad-serv.shtml#PLANT) de manera eficaz.

Estos contienen un conjunto de instrucciones que nos permiten realizar [operaciones](http://www.monografias.com/trabajos6/diop/diop.shtml) de entrada / salida, calculo, manipulación de textos, [lógica](http://www.monografias.com/trabajos15/logica-metodologia/logica-metodologia.shtml) / comparación y [almacenamiento](http://www.monografias.com/trabajos12/dispalm/dispalm.shtml) / recuperación.

Los lenguajes de programación se clasifican en:

Lenguaje Máquina: Son aquellos cuyas instrucciones son directamente entendibles por la computadora y no necesitan [traducción](http://www.monografias.com/trabajos32/traductor/traductor.shtml) posterior para que la [CPU](http://www.monografias.com/trabajos12/comptcn/comptcn.shtml#UCP) pueda comprender y ejecutar el [programa](http://www.monografias.com/Computacion/Programacion/). Las instrucciones en lenguaje maquina se expresan en términos de la unidad de [memoria](http://www.monografias.com/trabajos13/memor/memor.shtml) más pequeña el bit (dígito binario 0 ó 1).

Lenguaje de Bajo Nivel ([Ensamblador](http://www.monografias.com/trabajos14/lenguaje-ensamblador/lenguaje-ensamblador.shtml)): En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos para las operaciones y direcciones simbólicas.

Lenguaje de Alto Nivel: Los [lenguajes de programación](http://www.monografias.com/Computacion/Programacion/) de alto nivel (BASIC, [pascal](http://www.monografias.com/trabajos5/estat/estat.shtml), [cobol](http://www.monografias.com/trabajos/marcohistocomp/marcohistocomp.shtml), fortran, etc.) son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos (en general en inglés), lo que facilita la [escritura](http://www.monografias.com/trabajos16/metodo-lecto-escritura/metodo-lecto-escritura.shtml) y comprensión del programa.

[Algoritmo](http://www.monografias.com/trabajos15/algoritmos/algoritmos.shtml). La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhowarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y [ecuaciones](http://www.monografias.com/trabajos13/sumato/sumato.shtml#SOLUCION) en el siglo IX.

La solución a cualquier problema de cómputo involucra la ejecución de una serie de [acciones](http://www.monografias.com/trabajos4/acciones/acciones.shtml) en orden específico. Un [procedimiento](http://www.monografias.com/trabajos13/mapro/mapro.shtml) para resolver un problema en términos de: a) Las acciones a ejecutarse y b) el orden en el cual estas acciones deben ejecutarse se llama algoritmo.

Un ejemplo de un algoritmo para llegar a la escuela

1. Salir de la cama
2. Quitarse la pijamas
3. Darse un baño
4. Vestirse
5. Desayunar

Programa.

* + Secuencia de instrucciones mediante las cuales se ejecutan diferentes acciones de acuerdo con los [datos](http://www.monografias.com/trabajos11/basda/basda.shtml) que se estén procesando.
  + Es un algoritmo desarrollado para ser utilizado por la computadora

Expresión de un algoritmo en un lenguaje preciso que puede llegar a entender una máquina de cómputo.

1. Utilizar el [transporte](http://www.monografias.com/trabajos/transporte/transporte.shtml) ( autobús, carro , bicicleta, etc)

## Fases para la creación de un programa.

Definición del Problema

Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.

Análisis del Problema

Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:

Los datos de entrada. Cuál es la [información](http://www.monografias.com/trabajos7/sisinf/sisinf.shtml) que se desea producir (salida) Los [métodos](http://www.monografias.com/trabajos11/metods/metods.shtml) y fórmulas que se necesitan para procesar los datos. Una recomendación muy práctica es el que nos pongamos en el lugar de la computadora y analicemos que es lo que necesitamos que nos ordenen y en que secuencia para producir los resultados esperados.

[Diseño](http://www.monografias.com/trabajos13/diseprod/diseprod.shtml) del Algoritmo

Las características de un buen algoritmo son:

Debe tener un punto particular de inicio.

Debe ser definido, no debe permitir dobles interpretaciones.

Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.

Debe ser finito en tamaño y [tiempo](http://www.monografias.com/trabajos901/evolucion-historica-concepciones-tiempo/evolucion-historica-concepciones-tiempo.shtml) de ejecución.

Codificación

La [codificación](http://www.monografias.com/trabajos37/codificacion/codificacion.shtml) es la operación de escribir la solución del problema (de acuerdo a la lógica del [diagrama](http://www.monografias.com/trabajos14/flujograma/flujograma.shtml) de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un [código](http://www.monografias.com/trabajos12/eticaplic/eticaplic.shtml) reconocible por la computadora, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

Prueba y Depuración

Los errores humanos dentro de la programación de [computadoras](http://www.monografias.com/trabajos15/computadoras/computadoras.shtml) son muchos y aumentan considerablemente con la complejidad del problema. El [proceso](http://www.monografias.com/trabajos14/administ-procesos/administ-procesos.shtml#PROCE) de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración. La prueba consiste en la captura de datos hasta que el programa no presente errores (los más comunes son los sintácticos y lógicos).

Documentación

Es la guía o [comunicación](http://www.monografias.com/trabajos12/fundteo/fundteo.shtml) escrita es sus variadas formas, ya sea en enunciados, [procedimientos](http://www.monografias.com/trabajos13/mapro/mapro.shtml), [dibujos](http://www.monografias.com/trabajos13/histarte/histarte.shtml#ORIGEN) o [diagramas](http://www.monografias.com/trabajos12/diflu/diflu.shtml).

A menudo un programa escrito por una [persona](http://www.monografias.com/trabajos7/perde/perde.shtml), es usado por otra. Por ello la [documentación](http://www.monografias.com/trabajos11/ladocont/ladocont.shtml) sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones ([mantenimiento](http://www.monografias.com/trabajos15/mantenimiento-industrial/mantenimiento-industrial.shtml)).

La documentación se divide en tres partes:

Documentación Interna

Documentación Externa

Manual del Usuario

Documentación Interna: Son los comentarios o mensaje que se añaden al código fuente para hacer más claro el entendimiento de un proceso.

Documentación Externa: Se define en un documento escrito los siguientes puntos:

[Descripción](http://monografias.com/trabajos10/anali/anali.shtml) del Problema

Nombre del Autor

Algoritmo (diagrama de flujo o pseudocódigo)

Diccionario de Datos

Código Fuente (programa)

Manual del Usuario: Describe paso a paso la manera cómo funciona el programa, con el fin de que el usuario obtenga el resultado deseado.

Mantenimiento

Se lleva acabo después de terminado el programa, cuando se detecta que es necesario hacer algún [cambio](http://www.monografias.com/trabajos2/mercambiario/mercambiario.shtml), ajuste o complementación al programa para que siga trabajando de manera correcta. Para [poder](http://www.monografias.com/trabajos35/el-poder/el-poder.shtml) realizar este [trabajo](http://www.monografias.com/trabajos34/el-trabajo/el-trabajo.shtml) se requiere que el programa este correctamente documentado.

Programación Didáctica

El concepto de programación didáctica proviene del campo de la educación y se refiere al fenómeno mediante el cual los educadores de diferentes niveles y tipo programan u organizan la didáctica del hermoso proceso de enseñanza y aprendizaje. La programación didáctica es justamente poner de manera ordenada y significativa (es decir, con un sentido lógico) los conocimientos, las tareas y actividades a realizar, los objetivos a cumplir, los recursos a usar y otros datos. Todos ellos en conjunto permiten tener una visión más clara del proceso de enseñanza y analizar de mejor modo los resultados a medida que estos se vayan obteniendo.

Cuando hablamos de didáctica estamos haciendo referencia a aquel proceso que representa la enseñanza y el aprendizaje entendidos como dos fenómenos correlativos y necesarios uno de otro. A través de la didáctica (que en griego quiere decir justamente 'enseñar'), el profesional de la educación puede establecer lazos entre los diferentes tipos de conocimientos y buscar los mejores métodos para hacer de ellos un cúmulo de datos o informaciones accesibles y aprehendibles para el alumno. La didáctica es, en otras palabras, el modo mediante el cual el docente elabora estrategias que utilizará en el proceso de enseñanza para asegurar el aprendizaje de los individuos que cumplan el rol de educandos o estudiantes.

La programación didáctica es entonces el proceso mediante el cual se programa a largo, mediano y corto plazo el modo en que esos elementos didácticos serán estructurados, ordenados y organizados a lo largo del ciclo lectivo correspondiente. Por lo general, la programación abarca todo un ciclo lectivo que normalmente dura alrededor de nueve meses. Es por esto que una buena programación didáctica requiere tener en cuenta un sinfín de variables que puedan ajustarse a las circunstancias específicas del momento, de los alumnos, del propio docente, del establecimiento, etc.

Léxico y programación

La programación se rige por reglas y un conjunto más o menos reducido de órdenes, expresiones, instrucciones y comandos que tienden a asemejarse a una [lengua natural](https://es.wikipedia.org/wiki/Lengua_natural) acotada (en inglés); y que además tienen la particularidad de una reducida ambigüedad. Cuanto menos ambiguo es un lenguaje de programación, se dice, es más potente. Bajo esta premisa, y en el extremo, el lenguaje más potente existente es el binario, con ambigüedad nula (lo cual lleva a pensar así del lenguaje ensamblador)

En los [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguajes_de_programaci%C3%B3n) de alto nivel se distinguen diversos elementos entre los que se incluyen el léxico propio del lenguaje y las reglas semánticas y sintácticas.

Programas y algoritmos

Un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema. Un programa normalmente implementa (traduce a un lenguaje de programación concreto) uno o más algoritmos. Un algoritmo puede expresarse de distintas maneras: en forma gráfica, como un [diagrama de flujo](https://es.wikipedia.org/wiki/Diagrama_de_flujo), en forma de código como en [pseudocódigo](https://es.wikipedia.org/wiki/Pseudoc%C3%B3digo) o un lenguaje de programación, en forma explicativa.

Los programas suelen subdividirse en partes menores, llamadas módulos, de modo que la complejidad algorítmica de cada una de las partes sea menor que la del programa completo, lo cual ayuda al desarrollo del programa. Esta es una práctica muy utilizada y se conoce como "refino progresivo".

Según [Niklaus Wirth](https://es.wikipedia.org/wiki/Niklaus_Wirth" \o "Niklaus Wirth), un programa está formado por los [algoritmos](https://es.wikipedia.org/wiki/Algoritmo) y la [estructura de datos](https://es.wikipedia.org/wiki/Estructura_de_datos).

La programación puede seguir muchos enfoques, o [paradigmas](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n), es decir, diversas maneras de formular la resolución de un problema dado. Algunos de los principales paradigmas de la programación son:

* [Programación declarativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa)
* [Programación estructurada](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada)
* [Programación modular](https://es.wikipedia.org/wiki/Programaci%C3%B3n_modular)
* [Programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos)

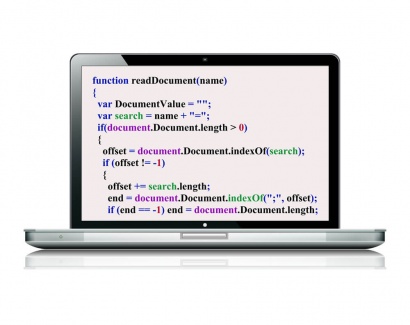
Código (programación / código abierto y cerrado)

Cuando hablamos de código de programación nos referimos el lenguaje por el cual funcionan las computadoras, comprende un conjunto de instrucciones y datos a ser procesados automáticamente.

El código en computación puede ser binario (interpretable solamente por computadoras), código fuente (interpretable por seres humanos), y en su aspecto legal o político puede ser software libre, código abierto, freeware, shareware o software privativo/propietario tradicional.

El software libre o Free Software posee una definición clara al respecto, según Free Software Foundation es aquel que puede ser usado con cualquier fin, estudiado (saber qué hace), copiado y mejorado. Existe una gran variedad de licencias de software libre, siendo la GNU GPL la referente de todas, pero podemos mencionar también las licencias MIT, BSD, Mozilla, Apache o Creative Commons.  
  
El software de código abierto (open source) es básicamente lo mismo que el software libre, salvo que no es reticente a mezclar software privativo con software libre. También existe software de código abierto que no puede ser utilizado con cualquier fin, por lo cual no sería libre.  
  
El freeware tiene poco que ver con el software libre, dado que lo único libre es su distribución: en general no puede ser modificado, estudiado o comercializado, por lo cual es obvio que viene sin código fuente (sin "la receta del programa").  
  
El shareware es similar al freeware pero con una limitación extra: el tiempo de uso. Estos programas suelen deshabilitarse en pocos días, son versiones demo/light, de programas completos privativos.  
  
El software privativo tradicional (al estilo de Windows) no permite su utilización con cualquier fin, no permite su estudio (salvo pagando fortunas a sus creadores), no permite su libre copia ni su mejora por parte de cualquier persona: es el inverso total del software libre, por eso Windows y GNU/Linux están tan enfrentados.

Código Fuente

Si le digo, amigo lector, que 01100011010100111100, usted puede interpretar que mi teclado está fallando, que he pulsado unas teclas al azar, que le quiero gastar una broma o que, simplemente, he perdido el juicio. No obstante, este código puede ser comprendido por una computadora, pues estas máquinas trabajan con código binario.  
  
Como para nosotros los seres humanos (incluso para el informático más avezado) es difícil mantener una “conversación” exclusivamente en binario con una computadora, se han creado formas intermedias de darles instrucciones a las computadoras que son más comprensibles que el lenguaje binario por parte de los seres humanos.

Una de estas formas intermedias es el código fuente en el que están escritos los programas informáticos.

Lenguajes de programación

Un lenguaje de programación es, en esencia, una forma de decirle a la computadora que es lo que queremos que haga, de una forma estructurada y comprensible por el ser humano (aunque para entenderlo del todo hace falta estudiar conceptos de programación y del lenguaje en concreto, tal como cuando se quiere aprender una nueva lengua).  
  
Existen varios lenguajes de programación, cada uno de ellos orientado a unas tareas o un público determinado.  
  
Los más conocidos son:  
  
BASIC.

Siglas de Beginner's All-purpose Symbolic Instruction Code (Código simbólico de instrucciones de propósito general para principiantes), fue inventado en 1964 como una forma de acercar la programación a los más profanos. Ligado inextricablemente a las computadoras de 8 bits que se popularizaron en la década de los 80 del siglo XX, inicialmente carecía de un control de ejecución estructurado, que fue incorporado en posteriores versiones del lenguaje.

C.

Si el BASIC es el lenguaje más popular, C es, sin lugar a dudas, el más influyente. Creado en 1972 por Dennis Ritchie con la portabilidad entre máquinas y arquitecturas distintas en mente, permitió reprogramar un embrionario sistema UNIX para que pudiera correr en distintas computadoras. Aún hoy es utilizado, en algunas variantes, para aplicaciones punteras en todos los campos.  
  
C++.

Una de las variantes antes mencionadas de C, tal vez la más influyente. Al lenguaje C le añade los conceptos de la orientación a objetos. Cobol. Desconocido para los profanos, es muy conocido entre los profesionales. Creado en 1959, la gran cantidad de soluciones creadas en la industria con él (especialmente en los sectores bancarios y de finanzas), han llevado a que todavía hoy sea utilizado, y su reemplazo se haga difícil.

PHP.

Orientado al trabajo con páginas web.

Java.

Creado por la extinta Sun Microsystems, su objetivo era ser realmente universal, programando sobre una sola máquina y ejecutando en cualquiera. Aunque solo cumplió este objetivo en parte y arrastra una serie de desventajas, también es aún muy utilizado.

Ensamblador.

En contra de los demás de la lista que son lenguajes de alto nivel, este es de bajo nivel, siendo el que más se acerca al código binario de la máquina aunque no es exactamente esto. Su código fuente es de muy difícil lectura para los profanos.

Lenguaje de programación

 Es el idioma utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

Hay muchos lenguajes de programación, pero para programar no es necesario conocer todos los lenguajes, es igual que cuando hablamos, podemos comunicarnos en español aunque no sepamos alemán. Aunque la palabra deberia ser idioma ya que lenguaje realmente abarca todos los idiomas pero en computación equivocadamente se usa el término lenguaje cuando el término correcto es idiomas de programación.

En la actualidad los lenguajes de programación estan escritos para ser comprensibles por el ser humano, a este código se le llama **código fuente**, pero no es comprendido por la máquina ya que esta solo maneja el lenguaje binario.

La compilación es el proceso de traducir un programa en código fuente a programa en código objeto (que usa el lenguaje binario), el programa encargado de compilar se llama compilador. La mayoria de software de programación trae su propio compilador.

### Tipos de lenguajes de programación

Los lenguajes de programación se clasifican en varios tipos, los hay por nivel, y por la forma como se ejecutan.

**Por nivel:** Hay lenguajes de programación de alto nivel y lenguajes de bajo nivel. Los lenguajes de alto nivel permiten que con pocas palabras se logre hacer lo mismo que se logra con un lenguaje de bajo nivel.

Ejemplos de lenguajes de alto nivel: C++, Python, Java

Ejemplos de lenguaje de bajo nivel: assembler

**Por la forma como se ejecutan** Hay lenguajes compilados e interpretados.

Los lenguajes compilados necesitan de un programa especial que lea el código fuente y cree un archivo binario ejecutable para una plataforma específica. Ejm: C++, Pascal.

Los lenguajes interpretados necesitan de un programa que traduzca en directo el código fuente escrito a instrucciones de la plataforma en la que se ejecutan. Ejm: Python, Visual Basic, Script.

Los primeros son más rápidos, mientras que los segundos son más lentos, esto debido a que al compilar un programa las órdenes son más entendibles para la computadora, mientraque al interpretarlo la máquina primero debe leer el código y convertir al paso las instrucciones a instrucciones de máquina entendibles para ella.

Entorno de Desarrollo Integrado

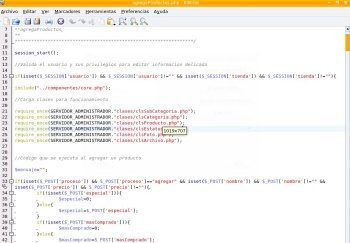
**Entorno de Desarrollo Integrado** (en inglés Integrated Development Environment 'IDE'): Es un programa compuesto por un conjunto de herramientas para un programador. Estos IDE son programas que sirven para programar, ya sea en un lenguaje de programación o en varios lenguajes. Los IDE que permiten crear programas en muchos lenguajes de programación permiten usar un solo programa para trabajar en varios lenguajes de programación, es decir no limitan al programador.

Ejemplos de IDES:

* Gambas (lenguaje derivado de BASIC),
* Eclipse (lenguaje Java),
* Kdevelop (varios lenguajes),
* Netbeans (varios lenguajes: java, php , C/C++),
* Visual Studio (varios lenguajes: C, C++, C#, Visual basic, ASP, Javascript)
* RAD Studio (Delphi, C++)

Compilación: hacerlo comprensible a la computadora

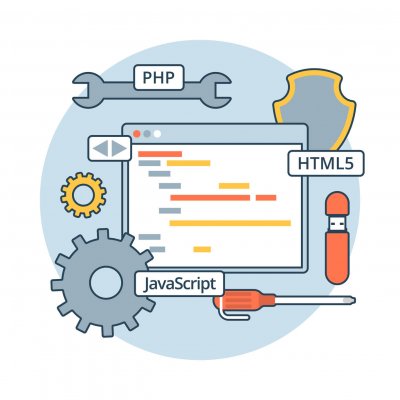
Cuando escribimos un programa en un lenguaje de programación como los anteriormente citados u otros, tenemos lo que se llama código fuente, el cual nosotros podemos entender con su simple lectura, pero no así la computadora.  
  
Para que esta última pueda entender lo que le decimos en el código fuente y ejecutar las instrucciones pertinentes, deberemos realizar un proceso que nos convertirá el código fuente en un fichero ejecutable. Dicho proceso se llama compilación.  
  
La compilación traduce el código fuente a código máquina.  
  
Por cierto, sobre el código del principio del artículo, aunque le he dicho que tal vez podría entenderlo una computadora, no me haga mucho caso; es simplemente un ejemplo que he escrito al azar, aunque el espíritu es este: las computadoras sólo “entienden” ceros y unos.  
  
Sintaxis

La sintaxis de programación es el conjunto de normas que regulan y coordinan las distintas variables y su asociación.

En la lengua, la sintaxis es el conjunto de normas y leyes combinatorias que estructuran la construcción de oraciones y textos. La sintaxis hace referencia a la inclusión de sujeto y predicado y al rol de las palabras en relación con otras. En las ciencias de la computación, la sintaxis comprende un concepto equivalente.

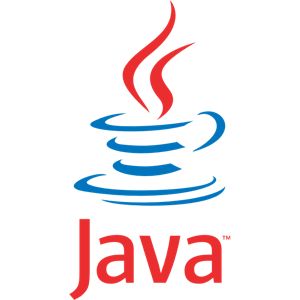
Para la informática, la sintaxis es un grupo de normas que funcionan gobernando la vinculación de las distintas variables que configuran las instrucciones operativas.  
  
En programación, existen tres variables relevantes: la sintaxis, la semántica y la jerarquía.  
La primera tiene que ver con que un lenguaje de programación de un software o aplicación puede entenderse como una serie de caracteres en particular en combinación. La sintaxis viene a estar compuesta de las reglas que determinan si dicha combinación o "string" es válida o no y, por lo tanto, operativa.  
  
Dentro de esa sintaxis también pueden encontrarse gramáticas y expresiones regulares. Esto quiere decir que existen caminos comunes que son usados a menudo por los programadores a la hora de trabajar asociando variables y caracteres.  
  
Términos comunes a la hora de hablar de sintaxis de programación son: los identificadores, las palabras reservadas, las literales o constantes y los símbolos especiales.  
  
Es común que en determinadas aplicaciones o programas recibamos un error clasificado por el mismo programa como "Error de sintaxis"; esto hace referencia a una falla en la programación de dicho software o en la combinación de las variables en su uso.  
  
Como usuarios, al ejecutar una operación o comando, probablemente veríamos un botón, menú u opción. Sin embargo, en el código de programación o 'backend' del programa o aplicación, se vería una sintaxis particular de asociación de caracteres.

JavaScript

Pese a que la coincidencia de nombre puede hacernos pensar en que este se trata de un derivativo del lenguaje de programación Java desarrollado por Sun Microsystems, esto no es así.  
  
JavaScript consiste en un lenguaje de programación interpretado, que habitualmente se utiliza en sitios web para ejecutar acciones en el lado del cliente, estando embebido en el código fuente de la página web  
  
Técnicamente, constituye un dialecto del estándar ECMAScript, propuesto por la entidad internacional de estándares de información y comunicación ECMA International y diseñado inicialmente por Netscape y, posteriormente, por la Fundación Mozilla. También constituye un estándar ISO.  
  
Si bien recibe en su sintaxis y forma de trabajo una mayor herencia del lenguaje C, también adopta nombres y convenciones que encontramos en Java, aunque se quedan en eso, en parecidos.  
  
Debido a su propósito y uso general, todos los navegadores web modernos interpretan correctamente JavaScript, siendo un lenguaje universal y multiplataforma. Y, habitualmente, su uso por parte de los desarrolladores se relaciona con la interfaz de usuario presentada por el sitio web.

JavaScript permite, en una página web, crear elementos como cuadros de diálogo, recoger información entrada por el usuario y mandarla al servidor para ser procesada

Al ser un lenguaje abierto y modificable en elementos como la sintaxis, se da a la creación de variantes y dialectos. Este es el caso de Microsoft, que dispone de su propia implementación bajo el nombre de JScript, el cual es utilizado tanto en el navegador Internet Explorer como en su sistema operativo bajo el nombre de Windows Scripting Host.  
  
Pese a que JavaScript se entiende, principalmente, desde el lado del cliente, también existen varias implementaciones en el lado del servidor que son ampliamente utilizadas. La principal de estas implementaciones para el lado del servidor es Node.js.  
  
Con el tiempo, JavaScript ha evolucionado, incluyendo funcionalidades de otros lenguajes más avanzados, como clases para programación orientada a objetos, o módulos para la reutilización de código fuente  
  
Pero pese a estos cambios y actualizaciones, JavaScript mantiene la compatibilidad hacia atrás:  
  
Es un lenguaje estructurado e imperativo, tomando como modelo la estructura de programación del lenguaje C  
Tipos dinámicos, de forma que cuando definimos el valor de la variable, se define el tipo de datos  
Orientado a objetos, con funciones que son consideradas como objetos  
Los objetos y métodos mediante los que el lenguaje puede interactuar con el usuario se encuentran limitados por el entorno de ejecución como, por ejemplo, el browser  
De igual forma que Perl, es compatible con expresiones regulares

Java

Java es un lenguaje de programación que fue desarrollado por la compañía “Sun Microsystems” que fue luego comprada por otra compañía llamada Oracle. El propósito de java fue el de crear un lenguaje con el que el internauta pudiese interactuar en internet y que además lo pudiese hacer, teniendo cualquier sistema operativo instalado. Sea Windows en cualquiera de sus versiones modernas, Mac, Linux, etc.  
  
Cualquiera de estos sistemas soporta java (en algunos sistemas viene incorporado, como en Mac. En otros hay que instalarlo, como en Windows). Java es especialmente útil cuando tienes que interactuar con un servidor en Internet como por ejemplo en los juegos online sencillos, tipo cartas, plataforma, aventuras etc. Normalmente los navegadores suelen llevar un complemento de java que permite ejecutar este programa a través de Internet Explorer, Chrome, Firefox, etc. etc. Uno aprieta una tecla y un monigote va hacia donde queramos. Existen también otros complementos para el navegador que permiten ejecutar juegos online pero no viene al caso ahora.

Java se suele usar en organismos gubernamentales para interactuar con documentación oficial. La flexibilidad de manejo y sencillez de java, permiten al servidor de una página web, hacer muchos cambios sin que el usuario se vea afectado o tenga que hacer grandes cambios en su manera interactuar con dicha página. Con solo bajarse la última versión de java de la página de Oracle ya puede interactuar con cualquier servidor online que exija java para el cumplimiento a través de la red de obligaciones documentales, certificados, pagos, facturas y un sinfín de gestiones.



## James Gosling cientifico nacido en Calgary (Canadá). El padre del este lenguaje de programación. Al ser tan sencillo y ser usado por casi 300 millones de usuarios, java es también objetivo de de ataques para conseguir el control de ordenadores y así chantajear al usuario para que pague para la recuperación de su configuración de trabajo con el ordenador. Es por este motivo importante una buena protección antivirus, tener siempre la última versión de java descargada en tu ordenador y borrar las versiones anteriores. También es importante tener la copia de seguridad de nuestros valiosos datos siempre actualizada. Reseñar que java es un tipo de código abierto y gratuito y que debido a los modernos compiladores o programadores automáticos que existen, es muy fácil programar y ejecutar en java. Su creador fue el científico James Gosling. Este "abandono" la compañía en Sun Microsistems en 2010. En referente a los motivos de su marcha alegó cambios radicales en la política de la nueva compañía. El lenguaje de programación Java

Java es una tecnología que hace sencilla la construcción de aplicaciones distribuidas, programas que son ejecutables por múltiples ordenadores a través de la red. En el estado del arte en la programación de red, Java promete extender el papel de Internet desde el terreno de las comunicaciones hacia una red en la cual puedan ejecutarse las aplicaciones completas. Su novedosa tecnología permitirá a los negocios proporcionar servicios de transacción a gran escala y en tiempo real y contener informacion interactiva en Internet. Java simplifica también la construcción de agentes software, programas que se mueven a través de la red y desempeñan funciones en ordenadores remotos en nombre del usuario. En un futuro cercano, los usuarios podrán enviar agentes software desde sus PCs hacia Internet para localizar información específica o para realizar transacciones en el menor tiempo posible en cualquier lugar del mundo.

Java llevará estos adelantos todavía más lejos haciendo posible suministrar aplicaciones completamente interactivas vía Web. Las razones por las cuales se ha prestado tanta atención al lenguaje Java podrían resumirse en la siguiente lista de posibilidades que Java ofrece a sus usuarios:

* Escribir programas fiables y robustos.
* Construir una aplicación para prácticamente cualquier plataforma, y ejecutar dicha aplicación en cualquier otra plataforma soportada sin tener que repetir la compilación del código.
* Distribuir sus aplicaciones a lo largo de la red de una forma segura.

En particular, los programas Java se pueden incrustar en documentos Web, convirtiendo páginas estáticas en aplicaciones que se ejecutan en el ordenador del usuario.

Como trabaja Java

Como muchos otros lenguajes de programación, Java utiliza un compilador para convertir el código fuente, legible para el ser humano, en programas ejecutables. Los compiladores tradicionales generan código que puede ejecutarse únicamente por un hardware específico. Los compiladores Java generan código binario o *bytecode* independiente de la arquitectura. Estos bytecodes se ejecutarán exclusivamente en una Máquina Virtual Java, *Virtual Machine*, *VM*, un procesador Java idealizado que normalmente se implementa por software, aunque la VM se ha implementado también como un chip hardware por Sun y otros.

Los archivos binarios Java se denominan archivos de clases debido a que contienen clases simples Java. Para ejecutar bytecodes, la máquina virtual utiliza cargadores de clases para obtener los bytecodes del disco o de la red. Cada archivo de clases se lleva entonces a un verificador de bytecodes que se asegura de que la clase tienen un formato correcto y que no llegará a corromper la memoria cuando se ejecute. Una vez verificados los bytecodes se interpretan por un intérprete.

Angular JS

AngularJS (comunalmente llamado "Angular.js" o "AngularJS 1"), es un *[framework](https://es.wikipedia.org/wiki/Framework" \o "Framework)* de [JavaScript](https://es.wikipedia.org/wiki/JavaScript) de [código abierto](https://es.wikipedia.org/wiki/C%C3%B3digo_abierto), mantenido por [Google](https://es.wikipedia.org/wiki/Google), que se utiliza para crear y mantener [aplicaciones web](https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_web) [de una sola página](https://es.wikipedia.org/wiki/Single-page_application). Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de [Modelo Vista Controlador](https://es.wikipedia.org/wiki/Modelo_Vista_Controlador) (MVC), en un esfuerzo para hacer que el desarrollo y las [pruebas](https://es.wikipedia.org/wiki/Pruebas_de_software) sean más fáciles.

La biblioteca lee el [HTML](https://es.wikipedia.org/wiki/HTML) que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript. Los valores de las variables de JavaScript se pueden configurar manualmente, o recuperados de los recursos [JSON](https://es.wikipedia.org/wiki/JSON) estáticos o dinámicos.

AngularJS se puede combinar con el entorno en tiempo de ejecución [Node.js](https://es.wikipedia.org/wiki/Node.js), el *framework* para servidor [Express.js](https://es.wikipedia.org/w/index.php?title=Express.js&action=edit&redlink=1) y la base de datos [MongoDB](https://es.wikipedia.org/wiki/MongoDB" \o "MongoDB) para formar el conjunto [MEAN](https://es.wikipedia.org/wiki/MEAN).

AngularJS está construido en torno a la creencia de que la [programación declarativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa) es la que debe utilizarse para generar [interfaces de usuario](https://es.wikipedia.org/wiki/Interfaz_de_usuario) y enlazar [componentes de software](https://es.wikipedia.org/wiki/Componente_de_software), mientras que la [programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa) es excelente para expresar la [lógica de negocio](https://es.wikipedia.org/wiki/L%C3%B3gica_de_negocio).[1](https://es.wikipedia.org/wiki/AngularJS#cite_note-1) Este framework adapta y amplía el HTML tradicional para servir mejor contenido dinámico a través de un [*data binding*](https://es.wikipedia.org/w/index.php?title=Data_binding&action=edit&redlink=1) bidireccional que permite la sincronización automática de modelos y vistas. Como resultado, AngularJS pone menos énfasis en la manipulación del [DOM](https://es.wikipedia.org/wiki/Document_Object_Model) y mejora la testeabilidad y el rendimiento.

Los objetivos de diseño:

* [Disociar](https://es.wikipedia.org/wiki/Separaci%C3%B3n_de_conceptos) la manipulación del DOM de la lógica de la aplicación. Esto mejora la capacidad de prueba del código.
* Considerar a las [pruebas](https://es.wikipedia.org/wiki/Pruebas_de_software) de la aplicación como iguales en importancia a la escritura de la aplicación. La dificultad de las pruebas se ve reducida drásticamente por la forma en que el código está estructurado.
* Disociar el lado del cliente de una aplicación del lado del servidor. Esto permite que el trabajo de desarrollo avance en paralelo, y permite la reutilización de ambos lados.
* Guiar a los desarrolladores a través de todo el proceso del desarrollo de una aplicación: desde el diseño de la [interfaz de usuario](https://es.wikipedia.org/wiki/Interfaz_de_usuario), a través de la escritura de la lógica del negocio, hasta las pruebas.

Angular sigue el patrón MVVM (Model View View-Model) de ingeniería de software y alienta la articulación flexible entre la presentación, datos y componentes lógicos. Con el uso de la inyección de dependencias, Angular lleva servicios tradicionales del lado del servidor, tales como controladores dependientes de la vista, a las aplicaciones web del lado del cliente. En consecuencia, gran parte de la carga en el backend se reduce, lo que conlleva a aplicaciones web mucho más ligeras.

Comandos básicos de SQL

## Definiendo cómo es almacenada la información.

* [CREATE DATABASE](https://mariadb.com/kb/en/create-database/) se utiliza para crear una nueva base de datos vacía.
* [DROP DATABASE](https://mariadb.com/kb/en/drop-database/) se utiliza para eliminar completamente una base de datos existente.
* [CREATE TABLE](https://mariadb.com/kb/en/create-table/) se utiliza para crear una nueva tabla, donde la información se almacena realmente.
* [ALTER TABLE](https://mariadb.com/kb/en/alter-table/) se utiliza para modificar una tabla ya existente.
* [DROP TABLE](https://mariadb.com/kb/en/drop-table/) se utiliza para eliminar por completo una tabla existente.

## Manipulando los datos.

* [SELECT](https://mariadb.com/kb/en/select/) se utiliza cuando quieres leer (o seleccionar) tus datos.
* [INSERT](https://mariadb.com/kb/en/insert/) se utiliza cuando quieres añadir (o insertar) nuevos datos.
* [UPDATE](https://mariadb.com/kb/en/update/) se utiliza cuando quieres cambiar (o actualizar) datos existentes.
* [DELETE](https://mariadb.com/kb/en/delete/) se utiliza cuando quieres eliminar (o borrar) datos existentes.
* [REPLACE](https://mariadb.com/kb/en/replace/) se utiliza cuando quieres añadir o cambiar (o reemplazar) datos nuevos o ya existentes.
* [TRUNCATE](https://mariadb.com/kb/en/truncate/) se utiliza cuando quieres vaciar (o borrar) todos los datos de la plantilla.

### Un ejemplo sencillo.

**CREATE** **DATABASE** mydb;

USE mydb;

**CREATE** **TABLE** mitabla ( id INT **PRIMARY** **KEY**, nombre VARCHAR(**20**) );

**INSERT** **INTO** mitabla **VALUES** ( **1**, 'Will' );

**INSERT** **INTO** mitabla **VALUES** ( **2**, 'Marry' );

**INSERT** **INTO** mitabla **VALUES** ( **3**, 'Dean' );

**SELECT** id, nombre **FROM** mitabla **WHERE** id = **1**;

**UPDATE** mitabla **SET** nombre = 'Willy' **WHERE** id = **1**;

**SELECT** id, nombre **FROM** mitabla;

**DELETE** **FROM** mitabla **WHERE** id = **1**;

**SELECT** id, nombre **FROM** mitabla;

**DROP** **DATABASE** mydb;

**SELECT** **count**(**1**) **from** mitabla; da el número de registros en la tabla

Booststrap

Bootstrap, es un framework originalmente creado por Twitter, que permite crear interfaces web con CSS y JavaScript, cuya particularidad es la de adaptar la interfaz del sitio web al tamaño del dispositivo en que se visualice. Es decir, el sitio web se adapta automáticamente al tamaño de una PC, una Tablet u otro dispositivo. Esta técnica de diseño y desarrollo se conoce como “**responsive design**” o diseño adaptativo.

El beneficio de usar responsive design en un sitio web, es principalmente que el sitio web se adapta automáticamente al dispositivo desde donde se acceda. Lo que se usa con más frecuencia, y que a mi opinión personal me gusta más, es el uso de *media queries*, que es un módulo de CSS3 que permite la representación de contenido para adaptarse a condiciones como la resolución de la pantalla y si trabajás las dimensiones de tu contenido en porcentajes, puedes tener una web muy fluida capaz de adaptarse a casi cualquier tamaño de forma automática.

Pero si no quieres nada que ver con los media queries, otra muy buena opción es el uso del framework de Bootstrap, que como te dije te ayudará a desarrollar tus sitios adaptativos.

Aun ofreciendo todas las posibilidades que ofrece Bootstrap a la hora de crear interfaces web, los diseños creados con Bootstrap son simples, limpios e intuitivos, esto les da agilidad a la hora de cargar y al adaptarse a otros dispositivos. El Framework trae varios elementos con estilos predefinidos fáciles de configurar: Botones, Menús desplegables, Formularios incluyendo todos sus elementos e integración jQuery para ofrecer ventanas y tooltips dinámicos.

Bootstrap tiene un soporte relativamente incompleto para [HTML5](http://es.wikipedia.org/wiki/HTML5) y [CSS 3](http://es.wikipedia.org/wiki/CSS_3), pero es compatible con la mayoría de los navegadores web. La información básica de compatibilidad de sitios web o aplicaciones está disponible para todos los dispositivos y navegadores. Existe un concepto de compatibilidad parcial que hace disponible la información básica de un sitio web para todos los dispositivos y navegadores. Por ejemplo, las propiedades introducidas en CSS3 para las esquinas redondeadas, gradientes y sombras son usadas por Bootstrap a pesar de la falta de soporte de navegadores antiguos. Esto extiende la funcionalidad de la herramienta, pero no es requerida para su uso.

Desde la versión 2.0 también soporta [diseños sensibles](http://es.wikipedia.org/w/index.php?title=Responsive_Web_Design&action=edit&redlink=1). Esto significa que el diseño gráfico de la página se ajusta dinámicamente, tomando en cuenta las características del dispositivo usado (Computadoras, tabletas, teléfonos móviles).

Eventos JQuery

Estos métodos se utilizan para registrar los comportamientos que se aplican cuando el usuario interactúa con el navegador, y para manipular aún más esos comportamientos registrados.

Enlazar

Adjuntar un manejador a un evento para los elementos.

.difuminar

Enlazar un controlador de eventos para el evento JavaScript “desenfoque”, o desencadenar ese evento en un elemento.

# [.cambio ()](https://api.jquery.com/change/)

Enlazar un controlador de eventos para el evento “cambio” JavaScript o desencadenar ese evento en un elemento.

# [Hacer clic ()](https://api.jquery.com/click/)

Enlazar un controlador de eventos para el evento “click” JavaScript o desencadenar ese evento en un elemento.

[.dblclick ()](https://api.jquery.com/dblclick/)

Enlazar un controlador de eventos para el evento JavaScript “dblclick”, o desencadenar ese evento en un elemento.

[jQuery.proxy ()](https://api.jquery.com/jQuery.proxy/)

Toma una función y devuelve una nueva que siempre tendrá un contexto particular.

[.tecla Arriba ()](https://api.jquery.com/keyup/)

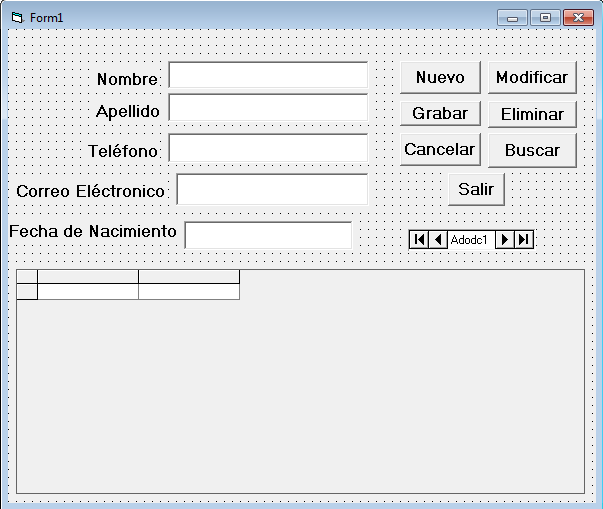
Enlazar un controlador de eventos al “keyUp” JavaScript evento, o desencadenar ese evento en un elemento.

Método CRUD

En [informática](https://es.wikipedia.org/wiki/Inform%C3%A1tica), CRUD es el [acrónimo](https://es.wikipedia.org/wiki/Acr%C3%B3nimo) de "Crear, Leer, Actualizar y Borrar" (del original en [inglés](https://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s): Create, Read, Update and Delete), que se usa para referirse a las funciones básicas en [bases de datos](https://es.wikipedia.org/wiki/Bases_de_datos) o la capa de persistencia en un [software](https://es.wikipedia.org/wiki/Software).

En algunos lugares, se utilizan las siglas ABM para lo mismo ("Alta, Baja y Modificación"), obviando la operación de "obtener"; el acrónimo ABC para "Altas, Bajas y Cambios"; ABML siendo la última letra (L) de "listar, listado o lectura"; ABMC siendo la 'C' de "Consulta"; o bien CLAB que sería la traducción literal del acrónimo ("Crear, Leer, Actualizar y Borrar").

También es usado el ABCDEF: "Agregar, Buscar, Cambiar, Desplegar (listar), Eliminar, Fichar (Ficha, cédula o Reporte de un registro)".



Metodo BOF

El bof es el primer el comienzo de un archivo y el eof pues es el final

Son practicos para recorer un record set... por ejemplo si tu record set se llama: rsTest entonces para evitar un error al recorrerlo escribes:  
  
If rsTest.BOF and rsTest.EOF Then  
... lo recorres...  
Else  
... no devolvio nada tu sentencia SQL...  
End If

Como sabrás, se puede usar .BOF o .EOF para comprobar si está en los límites del recordset.  
BOF devolverá True si el puntero está antes del primer registro, por otro lado EOF nos indicará si ese puntero está después del último registro, pero la recomendación para comprobar si realmente está vacío es que se cumplan esas dos condiciones: que tanto BOF como EOF sean True, por tanto podremos hacer una comprobación como esta:  
  
With rst  
If .EOF And .BOF Then  
lblData.Caption = "No hay ningún registro activo"  
.MoveFirst  
Else  
Text1(0) = .Fields("Au\_ID")  
' Por si el dato es nulo, añadirle una cadena vacia  
Text1(1) = .Fields("Author") & ""  
Text1(2) = .Fields("Year Born") & ""  
End If  
End With  
  
  
Aunque también se podría comprobar que .BOF Or .EOF sean True, es decir, hacer algo como esto:  
If .EOF Or .BOF Then  
pero ya te digo que lo más correcto es usar AND, al menos eso es lo que he leído en la documentación de ADO.

SCRUM

Scrum es un proceso en el que se aplican de manera regular [un conjunto de buenas prácticas](https://proyectosagiles.org/fundamentos-de-scrum) para**,** y obtener [el mejor resultado posible](https://proyectosagiles.org/beneficios-de-scrum) de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un [estudio de la manera de trabajar de equipos altamente productivos](https://proyectosagiles.org/historia-de-scrum).

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en **entornos complejos,** donde se necesita **obtener resultados pronto**, donde los**requisitos son cambiantes o poco definidos,** donde la**innovación,** la **competitividad,** la **flexibilidad** y la **productividad**son fundamentales.

Scrum también se utiliza para resolver situaciones en que **no se está entregando al cliente lo que necesita,** cuando**las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable,** cuando se necesita**capacidad de reacción ante la competencia,** cuando**la moral de los equipos es baja y la rotación alta**, cuando es necesario**identificar y solucionar ineficiencias sistemáticamente**o cuando se quiere trabajar utilizando un **proceso especializado en el desarrollo de producto.**

Ver en detalle cuales son los [beneficios de Scrum](https://proyectosagiles.org/beneficios-de-scrum), sus [fundamentos](https://proyectosagiles.org/fundamentos-de-scrum) y sus [requisitos](https://proyectosagiles.org/requisitos-de-scrum).

El proceso

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos ([iteraciones](https://proyectosagiles.org/desarrollo-iterativo-incremental) que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

El proceso parte de la [lista de objetivos/requisitos priorizada](https://proyectosagiles.org/lista-requisitos-priorizada-product-backlog) del producto, que actúa como plan del proyecto. En esta lista el [cliente](https://proyectosagiles.org/cliente-product-owner) prioriza los objetivos balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas.

Las actividades que se llevan a cabo en Scrum son las siguientes:

[Planificación de la iteración](https://proyectosagiles.org/planificacion-iteracion-sprint-planning)

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

1. Selección de requisitos (4 horas máximo). El cliente presenta al [equipo](https://proyectosagiles.org/equipo-team) la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
2. Planificación de la iteración (4 horas máximo). El equipo elabora la [lista de tareas de la iteración](https://proyectosagiles.org/lista-tareas-iteracion-sprint-backlog)necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se autoasignan las tareas.

[Ejecución de la iteración](https://proyectosagiles.org/ejecucion-iteracion-sprint)

Cada día el equipo realiza una [reunión de sincronización](https://proyectosagiles.org/reunion-diaria-de-sincronizacion-scrum-daily-meeting) (15 minutos máximo). Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso adquirido. En la reunión cada miembro del equipo responde a tres preguntas:

* ¿Qué he hecho desde la última reunión de sincronización?
* ¿Qué voy a hacer a partir de este momento?
* ¿Qué impedimentos tengo o voy a tener?

Durante la iteración el [Facilitador (Scrum Master)](https://proyectosagiles.org/facilitador-scrum-master) se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.

* Elimina los obstáculos que el equipo no puede resolver por sí mismo.
* Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

Durante la iteración, el cliente junto con el equipo [refinan la lista de requisitos (para prepararlos para las siguientes iteraciones) y, si es necesario, cambian o replanifican los objetivos del proyecto](https://proyectosagiles.org/replanificacion-proyecto) para [maximizar la utilidad de lo que se desarrolla](https://proyectosagiles.org/beneficios-de-scrum#flexibilidad-adaptacion) y el [retorno de inversión](https://proyectosagiles.org/beneficios-de-scrum#gestion-roi).

Inspección y adaptación

El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:

1. [Demostración](https://proyectosagiles.org/demostracion-requisitos-sprint-review) (4 horas máximo). El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, re planificando el proyecto.
2. [Retrospectiva](https://proyectosagiles.org/retrospectiva-sprint-retrospective) (4 horas máximo). El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de ir eliminando los obstáculos identificados.

Ver en detalle las diferentes actividades, responsabilidades y herramientas en [cómo funciona Scrum](https://proyectosagiles.org/como-funciona-scrum).

Scrum Master

Lidera al [equipo](https://proyectosagiles.org/equipo-team) llevando a cabo las siguientes responsabilidades:

Velar por que todos los participantes del proyecto sigan los [valores y principios ágiles](https://proyectosagiles.org/historia-de-scrum#manifiesto-agil), las[reglas y proceso de Scrum](https://proyectosagiles.org/como-funciona-scrum) y guiar la colaboración intraequipo y con el cliente de manera que las sinergias sean máximas. Esto implica:

* + Asegurar que exista una [lista de requisitos priorizada](https://proyectosagiles.org/lista-requisitos-priorizada-product-backlog) y que esté [preparada](https://proyectosagiles.org/replanificacion-proyecto) antes de la siguiente iteración.
  + Facilitar las reuniones de Scrum ([planificación de la iteración](https://proyectosagiles.org/planificacion-iteracion-sprint-planning), [reuniones diarias de sincronización del equipo](https://proyectosagiles.org/reunion-diaria-de-sincronizacion-scrum-daily-meeting), [demostración](https://proyectosagiles.org/demostracion-requisitos-sprint-review), [retrospectiva](https://proyectosagiles.org/retrospectiva-sprint-retrospective)), de manera que sean productivas y consigan sus objetivos.

Enseñar al equipo a auto gestionarse. No da respuestas, si no que guía al equipo con preguntas para que descubra por sí mismo una solución.

* Quitar los impedimentos que el equipo tiene en su camino para conseguir el objetivo de cada iteración (proporcionar un resultado útil al [cliente](https://proyectosagiles.org/cliente-product-owner) de la manera más efectiva) y poder finalizar el proyecto con éxito. Estos obstáculos se identifican de manera sistemática en las [reuniones diarias de sincronización del equipo](https://proyectosagiles.org/reunion-diaria-de-sincronizacion-scrum-daily-meeting) y en las reuniones de [retrospectiva](https://proyectosagiles.org/retrospectiva-sprint-retrospective).
* Proteger y aislar al equipo de interrupciones externas durante la [ejecución de la iteración](https://proyectosagiles.org/ejecucion-iteracion-sprint)(introducción de nuevos requisitos, "secuestro" no previsto de un miembro del equipo, etc.). De esta manera, el equipo puede mantener su productividad y el compromiso que adquirió sobre los requisitos que completaría en la iteración [notar, sin embargo, que el equipo debe reservar tiempo para colaborar con al cliente en la [preparación de la lista de requisitos para la próxima iteración](https://proyectosagiles.org/replanificacion-proyecto)].

### Metodologías de Programación

Programación estructurada

* Metodología descendente
* Metodología ascendente

La programación estructurada es una teoría de programación que consiste en construir programas de fácil comprensión, es especialmente útil, cuando se necesitan realizar correcciones o modificaciones después de haber concluido un programa o aplicación. Al utilizar la programación estructurada, es mucho más sencillo entender la codificación del programa, que se habrá hecho en diferentes secciones.

Se basa en una metodología de desarrollo de programas llamada refinamiento sucesivo: Se plantea una operación como un todo y se divide en segmentos más sencillos o de menor complejidad, una vez terminado todos los segmentos del programa, se procede a unificar las aplicaciones realizadas por el grupo de programadores. Si se ha utilizado adecuadamente la programación estructurada, esta integración debe ser sencilla y no presentar problemas al integrar la misma, y de presentar algún problema, será rápidamente detectable para su corrección.

La representación gráfica de la programación estructurada se realiza a través de diagramas de flujo, el cual representa el programa con sus entradas, procesos y salidas.

La programación estructurada propone segregar los procesos en estructuras lo más simple posibles, las cuales se conocen como secuencia, selección e interacción, que están disponibles en todos los lenguajes modernos de programación imperativa en forma de sentencias, combinando esquemas sencillos se pueden llegar a construir sistemas amplios y complejos pero de fácil entendimiento.

La programación estructurada es un método disciplinado de escribir programas que sean claros, que se demuestre que sean correctos y fáciles de modificar.

La programación estructurada consiste en dividir los programas en módulos y se basa en el desarrollo de programas que van de lo general a lo particular, es decir, del conjunto al elemento, es decir de un todo a lo específico.

Para la solución de un problema en particular, se inicia considerando las funciones que tiene que cumplir el programa en general y después se va desmembrando estas funciones en subfunciones más pequeñas hasta llegar al caso último o más particular y que ya no se pueda subdividir en casos más pequeños. Una vez que ya se tiene el programa desmembrado en de lo general a lo particular, se empieza a programar estas funciones pequeñas, particulares o módulos, de esta manera, siempre podremos construir nuevos módulos o unidades insertando el nombre del módulo donde corresponda y desarrollándolo a parte.

La modificación de los módulos es más fácil y se pueden referenciar cuantas veces se requiera, con lo que se ahorra tiempo en la programación, un programa tiene un diseño estructurado si cumple las dos siguientes condiciones:

* El teorema de Estructura.
* Está debidamente documentado

El teorema de Estructura dice que “un programa cumple el teorema de estructura si y sólo (ó) si es propio y contiene únicamente las tres estructuras básicas de control” que son la secuencial, la alternativa y la repetitiva, un programa es propio si y sólo si cumple: que tenga un solo punto de entrada y un solo punto de salida y que entre dos puntos de control del programa exista al menos un camino.

La programación estructurada es un estilo con el cual él se busca que el programador elabore programas sencillos y fáciles de entender, la programación estructurada hace uso de tres estructuras básicas de control que son: Estructura Secuencial, Estructura Selectiva y la Estructura Repetitiva (ó Iterativa)

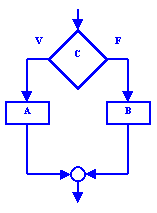
La programación estructurada se basa un teorema fundamental, el cual afirma que cualquier programa, no importa el tipo de trabajo que ejecute, puede ser elaborado utilizando únicamente las tres estructuras básicas.

**DEFINICIÓN DE LAS 3 ESTRUCTURAS BÁSICAS**

1. Estructura Secuencial: Indica que las instrucciones de un programa se ejecutan una después de la otra, en el mismo orden en el cual aparecen en el programa. Se representa gráficamente como una caja después de otra, ambas con una sola entrada y una única salida.

secuencia

Las cajas A y B pueden ser definidas para ejecutar desde una simple instrucción hasta un módulo o programa completo, siempre y cuando éstos también sean programas apropiados.

2. Estructura Selectiva: También conocida como la estructura si verdadero - falso, plantea la selección entre dos alternativas con base en el resultado de la evaluación de una condición; equivale a la instrucción IF de todos los lenguajes de programación y se representa gráficamente de la siguiente manera:

En el diagrama de flujo anterior, C es una condición que se evalúa; A es la acción que se ejecuta cuando la evaluación de esta condición resulta verdadera y B es la acción ejecutada cuando el resultado de la evaluación indica falso. La estructura también tiene una sola entrada y una sola salida; y las funciones A y B también pueden ser cualquier estructura básica o conjunto de estructuras.

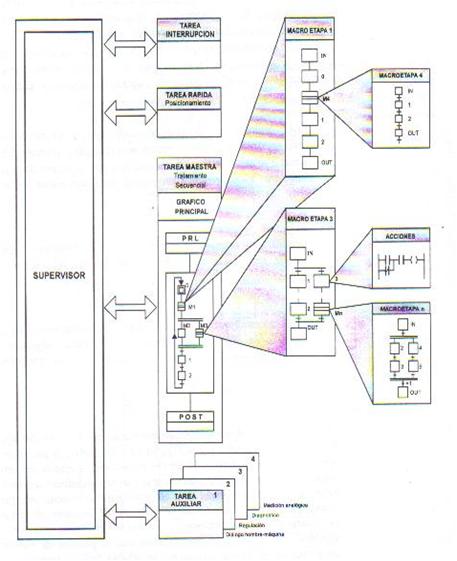
**VENTAJAS DE LA PROGRAMACIÓN ESTRUCTURADA**

Con la programación estructurada, elaborar programas de computadora sigue siendo una labor que demanda esfuerzo, creatividad, habilidad y cuidado. Sin embargo, con este nuevo estilo podemos obtener las siguientes ventajas:

1. Los programas son más fáciles de entender, un programa estructurado puede ser leído en secuencia, de arriba hacia abajo, sin necesidad de estar saltando de un sitio a otro en la lógica, lo cual es típico de otros estilos de programación.

2. Se logra una reducción del esfuerzo en las pruebas, el seguimiento de las fallas o depuración (debugging) se facilita debido a la lógica más visible, de tal forma que los errores se pueden detectar y corregir más fácilmente.

3. Se crean programas más sencillos y más rápidos.

****DIFERENCIAS CON LA PROGRAMACIÓN ESTRUCTURADA CON OTRO TIPO DE PROGRAMACIÓN

Aunque la programación estructurada condujo a mejoras de la técnica de programación secuencial, los métodos modernos de diseño de software orientado a objetos incluyen mejoras entre las que están el uso de los patrones de diseño, diseño por contrato, y lenguajes de modelado, Las principales diferencias entre la programación estructurada y la orientada a objetos son:

La programación orientada a objetos es más moderna, es una evolución de la programación estructurada que plasma en el diseño de una familia de lenguajes conceptos que existían previamente con algunos nuevos.

La programación orientada a objetos se basa en lenguajes que soportan sintáctica y semánticamente la unión entre los tipos abstractos de datos y sus operaciones.

* La programación orientada a objetos incorpora en su entorno de ejecución mecanismos tales como el polimorfismo y el envío de mensajes entre objetos.

Erróneamente se le adjudica a la programación estructurada clásica ciertos problemas como si fueran inherentes a la misma, esos problemas fueron haciéndose cada vez más graves y antes de la programación orientada a objetos diversos autores encontraron soluciones basadas en aplicar estrictas metodologías de trabajo. De esa época son los conceptos de cohesión y acoplamiento, de esos problemas se destacan los siguientes:

Modelo mental anómalo. Nuestra imagen del mundo se apoya en los seres, a los que asignamos nombres sustantivos, mientras la programación clásica se basa en el comportamiento, representado usualmente por verbos.

* Es difícil modificar y extender los programas, pues suele haber datos compartidos por varios subprogramas, que introducen interacciones ocultas entre ellos.
* Es difícil mantener los programas. Casi todos los sistemas informáticos grandes tienen errores ocultos, que no surgen a la luz hasta después de muchas horas de funcionamiento.
* Es difícil reutilizar los programas. Es prácticamente imposible aprovechar en una aplicación nueva las subrutinas que se diseñaron para otra.
* Es compleja la coordinación y organización entre programadores para la creación de aplicaciones de media y gran envergadura.

En la programación orientada a objetos pura no deben utilizarse llamadas de subrutinas, únicamente mensajes, por ello, a veces recibe el nombre de programación sin call, igual que la programación estructurada se llama también programación sin goto, sin embargo, no todos los lenguajes orientados a objetos prohíben la instrucción call, permitiendo realizar programación híbrida, imperativa y orientada a objetos a la vez.

Programación Orientada a Objetos

* Abstracción
* Encapsulamiento
* Herencia
* Polimorfismo

La programación Orientada a objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación. Con la POO tenemos que aprender a pensar las cosas de una manera distinta, para escribir nuestros programas en términos de objetos, propiedades, métodos y otras cosas que veremos rápidamente para aclarar conceptos y dar una pequeña base que permita soltarnos un poco con este tipo de programación.

## Motivación

Durante años, los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser utilizados por otras personas se creó la POO. Que es una serie de normas de realizar las cosas de manera que otras personas puedan utilizarlas y adelantar su trabajo, de manera que consigamos que el código se pueda reutilizar.

La POO no es difícil, pero es una manera especial de pensar, a veces subjetiva de quien la programa, de manera que la forma de hacer las cosas puede ser diferente según el programador. Aunque podamos hacer los programas de formas distintas, no todas ellas son correctas, lo difícil no es programar orientado a objetos sino programar bien. Programar bien es importante porque así nos podemos aprovechar de todas las ventajas de la POO.

## Cómo se piensa en objetos

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO. Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o aparcar.

Pues en un esquema POO el coche sería el objeto, las propiedades serían las características como el color o el modelo y los métodos serían las funcionalidades asociadas como ponerse en marcha o parar.

Por poner otro ejemplo vamos a ver cómo modelizaríamos en un esquema POO una fracción, es decir, esa estructura matemática que tiene un numerador y un denominador que divide al numerador, por ejemplo 3/2.

La fracción será el objeto y tendrá dos propiedades, el numerador y el denominador. Luego podría tener varios métodos como simplificarse, sumarse con otra fracción o número, restarse con otra fracción, etc.

Estos objetos se podrán utilizar en los programas, por ejemplo en un programa de matemáticas harás uso de objetos fracción y en un programa que gestione un taller de coches utilizarás objetos coche. Los programas Orientados a objetos utilizan muchos objetos para realizar las acciones que se desean realizar y ellos mismos también son objetos. Es decir, el taller de coches será un objeto que utilizará objetos coche, herramienta, mecánico, recambios, etc.

## Clases en POO

Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que la definición de un objeto es la clase. Cuando programamos un objeto y definimos sus características y funcionalidades en realidad lo que estamos haciendo es programar una clase. En los ejemplos anteriores en realidad hablábamos de las clases coche o fracción porque sólo estuvimos definiendo, aunque por encima, sus formas.

Propiedades en clases   
Las propiedades o atributos son las características de los objetos. Cuando definimos una propiedad normalmente especificamos su nombre y su tipo. Nos podemos hacer a la idea de que las propiedades son algo así como variables donde almacenamos datos relacionados con los objetos.

Métodos en las clases   
Son las funcionalidades asociadas a los objetos. Cuando estamos programando las clases las llamamos métodos. Los métodos son como funciones que están asociadas a un objeto.

## Objetos en POO

Los objetos son ejemplares de una clase cualquiera. Cuando creamos un ejemplar tenemos que especificar la clase a partir de la cual se creará. Esta acción de crear un objeto a partir de una clase se llama instanciar (que viene de una mala traducción de la palabra instace que en inglés significa ejemplar). Por ejemplo, un objeto de la clase fracción es por ejemplo 3/5. El concepto o definición de fracción sería la clase, pero cuando ya estamos hablando de una fracción en concreto 4/7, 8/1000 o cualquier otra, la llamamos objeto.

Para crear un objeto se tiene que escribir una instrucción especial que puede ser distinta dependiendo el lenguaje de programación que se emplee, pero será algo parecido a esto.

miCoche = new Coche()

Con la palabra new especificamos que se tiene que crear una instancia de la clase que sigue a continuación. Dentro de los paréntesis podríamos colocar parámetros con los que inicializar el objeto de la clase coche.

Estados en objetos   
Cuando tenemos un objeto sus propiedades toman valores. Por ejemplo, cuando tenemos un coche la propiedad color tomará un valor en concreto, como por ejemplo rojo o gris metalizado. El valor concreto de una propiedad de un objeto se llama estado.

Para acceder a un estado de un objeto para ver su valor o cambiarlo se utiliza el operador punto.

miCoche.color = rojo

El objeto es miCoche, luego colocamos el operador punto y por último el nombre e la propiedad a la que deseamos acceder. En este ejemplo estamos cambiando el valor del estado de la propiedad del objeto a rojo con una simple asignación.

Mensajes en objetos   
Un mensaje en un objeto es la acción de efectuar una llamada a un método. Por ejemplo, cuando le decimos a un objeto coche que se ponga en marcha estamos pasándole el mensaje ponte en marcha.

Para mandar mensajes a los objetos utilizamos el operador punto, seguido del método que deseamos invocar.

miCoche.ponerseEnMarcha()

En este ejemplo pasamos el mensaje ponerseEnMarcha(). Hay que colocar paréntesis igual que cualquier llamada a una función, dentro irían los parámetros.

## Otras cosas

Hay mucho todavía que conocer de la POO ya que sólo hemos hecho referencia a las cosas más básicas. También existen mecanismos como la herencia y el polimorfismo que son unas de las posibilidades más potentes de la POO.

La herencia sirve para crear objetos que incorporen propiedades y métodos de otros objetos. Así podremos construir unos objetos a partir de otros sin tener que reescribirlo todo. Puedes encontrar en DesarrolloWeb.com un [artículo completo dedicado a la Herencia](http://www.desarrolloweb.com/articulos/herencia-en-programacion-orientada-objetos.html).

El polimorfismo sirve para que no tengamos que preocuparnos sobre lo que estamos trabajando, y abstraernos para definir un código que sea compatible con objetos de varios tipos. Puedes acceder a otro [artículo para saber más sobre Polimorfismo](http://www.desarrolloweb.com/articulos/polimorfismo-programacion-orientada-objetos-concepto.html).

Son conceptos avanzados que cuesta explicar en las líneas de ese informe. No hay que olvidar que existen libros enteros dedicados a la POO y aquí solo pretendemos dar un repaso a algunas cosas para que os suenen cuando tengáis que poneros delante de ellas en los lenguajes de programación que debe conocer un desarrollador del web. Sin embargo, si quieres saber más también puedes continuar leyendo en DesarrolloWeb.com, en el manual de la [teoría de la Programación orientada a objetos](http://www.desarrolloweb.com/manuales/teoria-programacion-orientada-objetos.html).

## Ejemplo concreto de programación orientada a objetos

Para conseguir un ejemplo concreto de lo que es la programación orientada a objetos, podemos entrar en el [Manual de PHP 5](http://www.desarrolloweb.com/manuales/58/). Realmente este manual explica las características de orientación a objetos de PHP 5 y ofrece ejemplos concretos de creación de clases con características como herencia, polimorfismo, etc.

### Tipos de Datos Primitivos y Operadores

DATOS, TIPOS DE DATOS Y OPERACIONES PRIMITIVAS

El primer objetivo de toda computadora es el manejo de la información o datos. Estos datos pueden ser las cifras de ventas de un supermercado o las calificaciones de una clase. Un dato es la expresión general que describe los objetos con los cuales opera una computadora. La mayoría de las computadoras pueden trabajar con varios tipos (modos) de datos. Los algoritmos y los programas correspondientes operan sobre esos tipos de datos.

 La acción de las instrucciones ejecutables de las computadoras se refleja en cambios en los valores de las partidas de datos. Los datos de entrada se transforman por el programa, después de las etapas intermedias, en datos de salida.

 En el proceso de resolución de problemas el diseño de la estructura de datos es tan importante como el diseño del algoritmo y del programa que se basa en el mismo.

 Existen dos tipos de datos: simples (sin estructura) y compuestos (estructurados). Los datos estructurados son conjuntos de partidas de datos simples con relaciones definidas entre ellos.

 Los distintos tipos de datos se representan en diferentes formas en la computadora. A nivel de máquina, un dato es un conjunto o secuencia de bits (dígitos 0 o 1). Los lenguajes de alto nivel permiten basarse en abstracciones e ignorar los detalles de la representación interna. Aparece el concepto de tipo de datos, así como su representación. Los tipos de datos simples son los siguientes:

**Numéricos** (integer, real)

**Lógicos** (boolean)

**Carácter** (char, string)

Existen algunos lenguajes de programación que admiten otros tipos de datos; complejos, que permiten tratar los números complejos, y otros lenguajes  que  también permiten declarar y definir sus propios tipos de datos: enumerados (enumerated) y subrango (subrange).

**DATOS NUMERICOS**

El tipo numérico es el conjunto de los valores numéricos. Estos pueden representarse en dos formas distintas:

* Tipo numérico entero (integer).
* Tipo numérico real (real).

**Enteros:** el tipo entero es un subconjunto finito de los números enteros. Los enteros son números completos, no tienen componentes fraccionarios o decimales y pueden ser negativos o positivos. Ejemplos de números enteros son:

5                             6

-15                         4

20                           17

1340                      26

Los enteros se denominan en ocasiones números de punto o coma fija. Los números enteros máximos y mínimos de una computadora suelen ser  -32768 a +32767 -. Los números enteros fuera de este rango no suelen representar como entero, sino como reales, aunque existen excepciones enteros largos.

**Reales:** el tipo real consiste en un subconjunto de los números reales. Los números reales siempre tienen un punto decimal y pueden ser positivos o negativos. Un número real consta de un entero y una parte decimal. Los siguientes ejemplos son números reales:

0.08                       3739.41

3.7452                  -52.321

-8.12                     3.0

En aplicaciones científicas se requiere una representación especial para manejar números muy grandes, como la masa de la Tierra, o muy pequeños, como la masa de un electrón. Una computadora sólo puede representar un número fijo de dígitos. Este número puede variar de una máquina a otra, siendo ocho dígitos un número típico. Este límite provocará problemas para representar y almacenar números muy grandes como son los ya citados o los siguientes:

4867213432                        0.00000000387

Existe un tipo de representación denominado notación exponencial o científica y que se utiliza para números muy grandes o muy pequeños. Así.

                367520100000000000000

Se representa en notación científica descomponiéndolo en grupos de tres dígitos

                367         520         100         000         000         000         000

Y posteriormente en forma de potencias de 10

3.675201  x  1020

y de modo similar

.0000000000302579

Se representa como

3.02579  x  10-11

La representación en coma flotante es una generalización de notación científica. Obsérvese que las siguientes expresiones son equivalentes:

3.675201 x 1019 =  .367507 x 1020  =   .03675201  x 1021  =   . . .

= 36.75201 x  1018 =  367.5201 x 1017=  . . .

En estas expresiones se considera la mantisa (parte decimal) al número real y el exponente (parte potencial) el de la potencia de diez.

                36.75201              mantisa                                               18           exponente

**DATOS LOGICOS**

El tipo lógico –también denominado booleano- es aquel dato que sólo puede tomar uno de dos valores:

                Cierto   o             verdadero          (true)    y   falso                (false).

Este tipo de datos se utiliza para representar las alternativas (si/no) a determinadas condiciones. Por ejemplo, cuando se pide si un valor entero es par, la respuesta será verdadera o falsa, según sea par o impar.

**DATOS TIPO  CARÁCTER Y TIPO CADENA**

El tipo carácter es el conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato tipo carácter contiene un solo carácter. Los caracteres que reconocen las diferentes computadoras no son estándar; sin embargo, la mayoría reconoce los siguientes caracteres alfabéticos y numéricos:

* Caracteres alfabéticos (A, B, C, . . . , Z)  (a, b, c, . . . , z).
* Caracteres numéricos (1, 2,. . ., 9, 0).
* Caracteres especiales (+, -, \*, /, ^, <, >, $, . . .).

Una cadena (string) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla (apóstrofo) o dobles comillas, según el tipo de lenguaje de programación. La longitud de una cadena de caracteres es el número de ellos comprendidos entre los separadores o limitadores. Algunos lenguajes tienen datos tipo cadena.

                ‘Hola Mortimer ’

                ‘12 de octubre de 1492 ‘

                ‘Sr. McKoy ‘

**CONSTANTES Y VARIABLES**

Los programas de computadora contienen ciertos valores que no deben cambiar durante la ejecución del programa. Tales valores se llaman constantes. De igual forma, existen otros valores que cambiarán durante la ejecución del programa; a estos valores se les llama variables. Una constante es una partida de datos (objetos) que permanecen sin cambios durante todo el desarrollo del algoritmo o durante la ejecución del programa.

Pi = 3.141592                     o                             E= 2.718228

***Constantes reales válidas***

1.234

-0.1436

+ 54437324

***Constantes reales no válidas***

1.753.63 (comas no permitidas)

82 (normalmente contienen un punto decimal, aunque existen lenguajes que lo admiten sin punto)

***Constantes reales en notación científica***

3. 7456 E2  equivale a   3.374562  x 102

Una constante tipo carácter o constante de caracteres consiste en un carácter válido encerrado dentro de apóstrofos; por ejemplo,

                ‘B’          ‘+’          ‘4’           ‘;’

Si se desea incluir el apóstrofo en la cadena, entonces debe aparecer como un par de apostrofes, encerrados dentro de simples comillas.

Una cadena de caracteres se denomina normalmente una cadena y una constante tipo cadena es una cadena encerrada entre apóstrofes. Por consiguiente,

                ‘Juan Manrique’              y             ‘Pepe Luis Aguilar’

Son constantes de cadena válidas. Nuevamente, si un apóstrofe es uno de los caracteres es una constante de cadena, debe aparecer como un par de apóstrofes.

                ‘John’’s’

**Constantes lógicas (boolean)**

Sólo existen dos constantes lógicas o boolean:

                Verdadero         falso

La mayoría de lenguajes de programación permiten diferentes tipos de constantes: entereas, reales, caracteres y booleanas o lógicas, y representan datos de esos tipos.

Una variable es un objeto o partida de datos cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución  del programa. Dependiendo del lenguaje, hay diferentes tipos de variables, tales como enteras, reales, carácter, lógicas y de cadena. Una variable que es de un cierto tipo puede tomar únicamente valores de ese tipo. Una variable de carácter, por ejemplo, puede tomar como valor sólo caracteres, mientras que una variable entera puede tomar sólo valores enteros.

Si se intenta asignar un valor de un tipo de una variable de otro tipo se producirá un error de tipo. Una variable se identifica por los siguientes atributos: nombre que lo asigna y tipo que describe el uso de la variable.

Los nombres de las variables, a veces conocidos como identificadores, suelen constar de varios caracteres  alfanuméricos, de los cuales el primero normalmente es una letra. No se deben utilizar –aunque lo permita el lenguaje- como nombres de identificadores palabras reservadas del lenguaje de programación. Nombres válidos de variables son:

A510

NOMBRES

NOTAS

NOMBRE\_APELLIDOS

Los nombres de las variables elegidas para el algoritmo o el programa deben ser significativos y tener relación con el objeto que representan, como pueden ser los casos siguientes:

                NOMBRE             para representar nombres de personas

                PRECIOS              para representar los precios de diferentes artículos

                NOTAS                 para representar las notas de una clase

**EXPRESIONES**

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Las mismas ideas son utilizadas en notación matemática tradicional; por ejemplo:

Aquí los paréntesis indican el orden de cálculo y representa la función raíz cuadrada.

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas. Una expresión consta de operandos y operadores. Según sea el tipo de objetos que manipulan, las expresiones se clasifican en:

* Aritméticas,
* Relacionales,
* Lógicas,
* Carácter.

El resultado de la expresión aritmética es de tipo numérico; el resultado de la expresión relacional y de una expresión lógica es de tipo lógico; el resultado de una expresión carácter es de tipo carácter.

**EXPRESIONES ARITMETICAS**

Las expresiones aritméticas son análogas a las fórmulas matemáticas. Las variables y constantes son numéricas (real o entera) y las operaciones son las aritméticas

  +               Suma

–                              resta

\*                             Multiplicación

/                             División

↑, \*\*, ˄              exponenciación

Div                         división entera

mod                      módulo (resto)

Los símbolos +,  -, \*, ˄ (↑ o \*\*) y las palabras clave div y mod se conocen como operadores aritméticos. En la expresión  
                5 + 3

Los valores 5 y 3 se denominan operandos. El valor de la expresión 5 + 3  se conoce como resultado de la expresión.

Los operadores se utilizan de igual forma que en matemáticas. Por consiguiente, A    x    B se escribe en un algoritmo como A \* B   y 1/ 4  x  C   como   C/4. Al igual que en matemáticas el signo menos juega un doble papel, como resta en  A   –  B y como negación en   –A.

                5  x   7                   se representa por           5 \*  7

                                             Se representa por           6/4

                37                           se representa por           3˄7

**EXPRESIONES LOGICAS (BOOLEANAS)**

Un segundo tipo de expresiones es la expresión lógica o booleana, cuyo valor es siempre verdadero o falso. Recuerde que existen dos constantes lógicas, verdadera (true) y falsa (false) y que las variables lógicas pueden tomar sólo estos dos valores. En esencia, una expresión lógica es una expresión que sólo puede tomar estos dos valores, verdadero y falso. Se denominan también expresiones booleanas en honor del matemático británico George Boole, que desarrolló el Álgebra lógica de Boole.

Las expresiones lógicas se forman combinando constantes lógicas, variables lógicas y otras expresiones lógicas, utilizando los operadores lógicos not, and y or y los operadores relacionales (de relación o comparación) =, <, >, <=, >=, <>.

**OPERADORES DE RELACIÓN**

Los operadores relacionales o de relación permiten realizar comparaciones de valores de tipo numérico o carácter. Los operadores de relación sirven para expresar las condiciones en los algoritmos. Los operadores de relación se recogen en la tabla:

|  |  |
| --- | --- |
| **Operador** | **Significado** |
| **<**  **>**  **=**  **<=**  **>=**  **<>** | Menor que  Mayor que  Igual que  Menor o igual que  Mayor o igual que  Distinto de |

Algunos ejemplos de aplicación:

El resultado de una operación podría ser verdadero o falso. Así, por ejemplo, en      A  > B, si A= 4 y B = 3, entonces

                A > B                                     es verdadero.

Mientras que

                (A – 2)  <  (B – 4)              es falso.

Los operadores de relación se pueden aplicar a cualquiera de los cuatro tipos de datos estándar: enteros, real, lógico, carácter. La aplicación a valores numéricos es evidente.

**OPERADORES LOGICOS**

Los operadores lógicos o booleanos básicos son    **NOT (no), AND (y)  y OR  (o).**Esta tabla muestra su funcionamiento:

|  |  |  |
| --- | --- | --- |
| **Operador** | **Expresión lógica** | **Significado** |
| **no (not)**  **y (and)**  **o (or)** | no p (not p)  p y q (p and q)  p o q (p or q) | negación de p.  conjunción de p y q.  disyunción de p y q. |

Las definiciones de las operaciones no, y, o se resumen en unas tablas conocidas como tablas de verdad.

|  |  |
| --- | --- |
| a | no a |
| verdadero  falso | falso  verdadero |

 No (6>10) es verdadera ya que (6>10) es falsa.

|  |  |  |
| --- | --- | --- |
| a | b | a y b |
| verdadero  verdadero  falso  falso | verdadero  falso  verdadero  falso | verdadero  falso  falso  falso |

 A y b es verdadera sólo si a y b son verdaderas.

|  |  |  |
| --- | --- | --- |
| a | b | a o b |
| verdadero  verdadero  falso  falso | verdadero  falso  verdadero  falso | verdadero  verdadero  verdadero  falso |

 A o b es verdadera cuando a, b o ambas son verdaderas.

En las expresiones lógicas se pueden mezclar operadores de relación y lógicos. Así, por ejemplo,

(1 < 5) Y (5 < 10) es verdadera

(5 > 10) O (‘A’ < ‘B’) es verdadera, ya que ‘A’ < ‘B’

# Estructuras de control: IF, FOR, FOR EACH, WHILE

Las estructuras de control nos permiten controlar el flujo del programa: tomar decisiones, realizar acciones repetitivas etc, dependiendo de unas condiciones que nosotros mismos establezcamos. Así podemos hacer un script que nos salude cada día de la semana de una manera diferente. O por ejemplo hacer un script que nos pida la contraseña una y otra vez hasta que suministremos la opción correcta.

**1.- La toma de decisiones if-else:**

Quizá la estructura de control más básica y la más utilizada sea if-else:

if (condicion) {  
// instrucciones que hay que ejecutar si la condición es verdadera  
} else {  
// Instrucciones que hay que ejecutar si la condición es falsa  
}  
En castellano sería algo así:

Si (Tal cosa es cierta) {  
//  Haz esto  
} En caso contrario {  
// Haz esto otro  
}  
Veamos un ejemplo real: En un formulario preguntaremos al usuario como se llama, pasaremos las variables a la página “saludos.php”. En saludos.php mostraremos un saludo diferente en función de si el usuario soy yo o es cualquier otro

Formulario

<html>  
<head>  
<title> Ejemplo básico de If </title>  
</head>  
<body>

<form action="saludos.php" method="get">

**¿Quién eres?**<br />  
<input type="text" name="nombre" />

<input type="submit" name="reset" value=" Enviar " />  
</form>  
</body>  
</html>

**saludos.php**

<html>

<head>  
<title> Ejemplo básico de If </title>  
</head>  
<body>  
<h1> Página de bienvenida</h1>  
<?php // Empieza el código php  
if ($nombre=="Coder") {  // Si el nombre es Coder  
echo(" Hombre $nombre! Ya estás cotilleando por aquí otra vez!"); // saludo especial  
}else {  
echo("Hola, $nombre, gracias por visitarnos"); // saludo para los demás  
}  
?>  
</body>  
</html>  
  
La cláusula else (la parte del if-else que indica lo que hay que hacer si la condición es falsa) es opcional. No es necesario ponerla. Por ejemplo puedo querer un script que sólo me salude a mí. En este caso, cambiaría saludos.php de la siguiente manera:

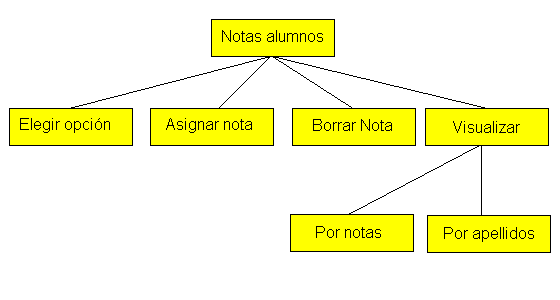
<html>  
<head>  
<title> Ejemplo básico de If </title>  
</head>  
<body>  
<h1> Página de bienvenida</h1>  
<?php // empieza el código php  
if ($nombre=="Coder") { // Si el nombre es Coder  
echo(" Hombre $nombre! Ya estás cotilleando por aquí otra vez!"); // Saludo para Coder  
}  
?>  
</body>  
</html>

**¿Qué es programación modular?**

Uno de los métodos más conocidos para resolver un problema es dividirlo en problemas más pequeños, llamados subproblemas. De esta manera, en lugar de resolver una tarea compleja y tediosa, resolvemos otras más sencillas y a partir de ellas llegamos a la solución. Esta técnica se usa mucho en programación ya que programar no es más que resolver problemas, y se le suele llamar diseño descendente, metodología del divide y vencerás o programación top-down.

Es evidente que si esta metodología nos lleva a tratar con subproblemas, entonces también tengamos la necesidad de poder crear y trabajar con subprogramas para resolverlos. A estos subprogramas se les suele llamar módulos, de ahí viene el nombre de programación modular. En Pascal disponemos de dos tipos de módulos: los [procedimientos](http://teleformacion.edu.aytolacoruna.es/PASCAL/document/modular.htm#procedimientos) y las [funciones](http://teleformacion.edu.aytolacoruna.es/PASCAL/document/modular.htm#funciones).

Veamos un ejemplo de cómo emplear el diseño descendente para resolver un problema. Supongamos que un profesor quiere crear un programa para gestionar las notas de sus alumnos. Quiere que dicho programa le permita realizar tareas tales como asignar notas, cambiar notas, ver las notas según distintas calificaciones, etc. A continuación tines un esquema que representa una de las posibles divisiones del problema en módulos.



**Los procedimientos**

Un procedimiento es un subprograma que realiza una tarea específica. Para invocarlo, es decir, para hacer que se ejecute, basta con escribir su nombre en el cuerpo de otro procedimiento o en el programa principal. Pero, hay que tener muy en cuenta que su [declaración](http://teleformacion.edu.aytolacoruna.es/PASCAL/document/estruct.htm#subprog) debe hacerse antes de que sea llamado por otro módulo.

Una vez que has construido varios programillas en Pascal, crear un procedimiento no es nada complicado, pues tiene prácticamente la misma estructura que un programa. Veamos las secciones que comparten y no comparten un procedimiento y un programa principal:

* Mientras que en el programa la cabecera consta de la palabra reservada program seguida del nombre del programa, en un procedimiento se compone de la palabra procedure seguida del nombre del procedimiento y una lista de parámetros que es opcional.
* Las secciones de declaración de constantes ([const](http://teleformacion.edu.aytolacoruna.es/PASCAL/document/estruct.htm" \l "ctes)), de tipos ([type](http://teleformacion.edu.aytolacoruna.es/PASCAL/document/estruct.htm" \l "tipos)) y de variables ([var](http://teleformacion.edu.aytolacoruna.es/PASCAL/document/estruct.htm" \l "vars)) también pueden aparecer en la estructura de cualquier procedimiento.
* Respecto al cuerpo del procedimiento, decir que al igual que el de un programa se delimita por las palabras reservadas begin y end, y en su interior puede contener sentencias [simples](http://teleformacion.edu.aytolacoruna.es/PASCAL/document/senten.htm#simples) o [estructuradas](http://teleformacion.edu.aytolacoruna.es/PASCAL/document/flujo.htm).
* Por último, comentar que ambos difieren en el signo de puntuación que marca su final, ya que en un programa es el punto y en un procedimiento es el punto y coma.

Todas estas diferencias y similitudes que hemos comentado, puedes apreciarlas en los siguientes esquemas que representan las estructuras de un programa y de un procedimiento:

Técnicas básicas de compilación

El análisis estático de programas juega un papel cada vez más importante en los compiladores actuales, ya que permite obtener información en tiempo de compilación sobre el comportamiento del programa en tiempo de ejecución, Dicha información es útil posteriormente tanto para comprobar que el programa cumple las especificaciones dadas como para optimizar el programa.

En esta tesis se presentan primero algunas técnicas básicas de análisis globales tales como algoritmos para análisis incremental eficiente, que permiten la reutilización de la información de análisis anteriores cuando un programa analizado sufre alguna modificación, y métodos para el análisis de las características complejas de los lenguajes de programación reales, tales como efectos laterales o modificación dinámica del programa.

También se proponen técnicas de optimización de programas, las cuales se encuadran dentro del marco general de la especialización: el programa resultante es válido para el caso particular considerado y más eficiente que el programa original. Se presenta un tipo de especialización que es abstracta, en el sentido de que el programa se especializa no con respecto a valores concretos, sino con respecto a valores abstractos, que también se pueden ver como conjuntos (posiblemente infinitos) de valores concretos, e profundiza en la relación entre la especialización abstracta y la evaluación parcial tradicional y en cómo integrar ambas en un nuevo marco que reúne las ventajas de cada una de ellas. Se presenta también un lenguaje de aserciones que permite al mismo tiempo al usuario expresar requisitos (especificaciones) sobre el programa y al analizador expresar los resultados del análisis. Esto facilita la comunicación entre el usuario y las distintas herramientas del entorno de desarrollo de programas, así como entre dichas herramientas. Finalmente, se presenta un esquema de validación de aserciones que comb

Conclusión