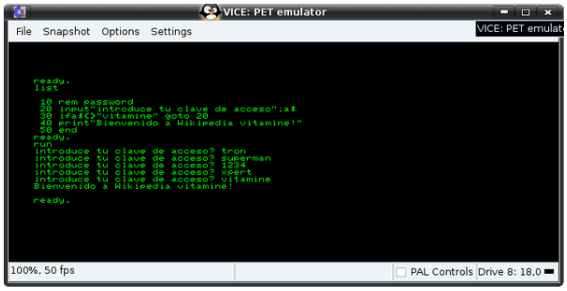
Lenguaje de programación



Captura de la microcomputadora Commodore PET-32 mostrando un programa en el lenguaje de programación BASIC, bajo el emulador VICE en una distribución GNU/Linux.

```java
// class declaration
public class ProgrammingExample {

    // method declaration
    public void sayHello() {

        // method output
        System.out.println("Hello World!");
    }
}
```

Un ejemplo de código fuente escrito en el lenguaje de programación Java, que imprimirá el mensaje "Hello World!" a la salida estándar cuando es compilado y ejecutado

Un **lenguaje de programación** es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como las computadoras.

Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.[1]

Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación.

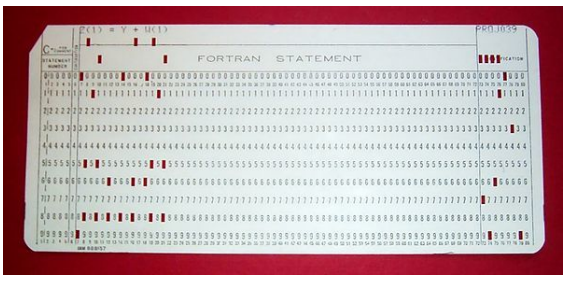
También la palabra programación se define como el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos, a través de los siguientes pasos:

- El desarrollo lógico del programa para resolver un problema en particular.
- Escritura de la lógica del programa empleando un lenguaje de programación específico (codificación del programa).
- Ensamblaje o compilación del programa hasta convertirlo en lenguaje de máquina.
- Prueba y depuración del programa.
- Desarrollo de la documentación.

Existe un error común que trata por sinónimos los términos 'lenguaje de programación' y 'lenguaje informático'. Los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como por ejemplo HTML (lenguaje para el marcado de páginas web que no es propiamente un lenguaje de programación, sino un conjunto de instrucciones que permiten estructurar el contenido de los documentos).

Permite especificar de *manera precisa* sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar *relativamente* próximo al lenguaje humano o natural. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa.

1 Historia



Código Fortran en una tarjeta perforada, mostrando el uso especializado de las columnas 1-5, 6 y 73-80.

Para que la computadora entienda nuestras instrucciones debe usarse un lenguaje específico conocido como código máquina, el cual la máquina comprende fácilmente, pero que lo hace excesivamente complicado para las personas.

De hecho sólo consiste en cadenas extensas de números 0 y 1.

Para facilitar el trabajo, los primeros operadores de computadoras decidieron hacer un traductor para reemplazar los 0 y 1 por palabras o abstracción de palabras y letras provenientes del inglés; éste se conoce como lenguaje ensamblador. Por ejemplo, para sumar se usa la letra A de la palabra inglesa *add* (sumar). El lenguaje ensamblador sigue la misma estructura del lenguaje máquina, pero las letras y palabras son más fáciles de recordar y entender que los números.

La necesidad de recordar secuencias de programación para las acciones usuales llevó a denominarlas con nombres fáciles de memorizar y asociar: ADD (sumar), SUB (restar), MUL (multiplicar), CALL (ejecutar subrutina), etc. A esta secuencia de posiciones se le denominó "instrucciones", y a este conjunto de instrucciones se le llamó lenguaje ensamblador. Posteriormente aparecieron diferentes lenguajes de programación, los cuales reciben su denominación porque tienen una estructura sintáctica similar a los lenguajes escritos por los humanos, denominados también lenguajes de alto nivel.

La primera programadora de computadora conocida fue Ada Lovelace, hija de Anabella Milbanke Byron y Lord Byron. Anabella introdujo en las matemáticas a Ada quien, después de conocer a Charles Babbage, tradujo y amplió una descripción de su máquina analítica. Incluso aunque Babbage nunca completó la construcción de cualquiera de sus máquinas, el trabajo que Ada realizó con éstas le hizo ganarse el título de primera programadora de computadoras del mundo. El nombre del lenguaje de programación Ada fue escogido como homenaje a esta programadora.

A finales de 1953, John Backus sometió una propuesta a sus superiores en IBM para desarrollar una alternativa más práctica al lenguaje ensamblador para programar la computadora central IBM 704. El histórico equipo Fortran de Backus consistió en los programadores Richard Goldberg, Sheldon F. Best, Harlan Herrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Lois Haibt y David Sayre.[2]

El primer manual para el lenguaje Fortran apareció en octubre de 1956, con el primer compilador Fortran entregado en abril de 1957. Esto era un compilador optimizado, porque los clientes eran reacios a usar un lenguaje de alto nivel a menos que su compilador pudiera generar código cuyo desempeño fuera comparable al de un código hecho a mano en lenguaje ensamblador.

En 1960, se creó COBOL, uno de los lenguajes usados aún en la actualidad, en informática de gestión.

A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método más eficiente para programarlas. Entonces, se crearon los lenguajes de alto nivel, como lo fue BASIC en las versiones introducidas en los microordenadores de la década de 1980. Mientras que una tarea tan sencilla como sumar dos números puede necesitar varias instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará una sola sentencia.

2 Elementos

2.1 Variables y vectores

Las variables podrían calificarse como contenedores de datos y por ello se diferencian según el tipo de dato que son capaces de almacenar. En la mayoría de lenguajes de programación se requiere especificar un tipo de variable concreto para guardar un dato concreto. Por ejemplo, en Java, si deseamos guardar una cadena de texto deberemos especificar que la variable es del tipo *String*. Por otra parte, en lenguajes como el PHP este tipo de especificación de variables no es necesario. Además, existen variables compuestas por varias variables llamadas vectores. Un vector no es más que un conjunto de variables consecutivas en memoria y del mismo tipo guardadas dentro de una variable contenedor. A continuación, un listado con los tipos de variables y vectores más comunes:

- Variables tipo Char: Estas variables contienen un único carácter, es decir, una letra, un signo o un número.
- Variables tipo Int: Contienen un número entero.
- Variables tipo float: Contienen un número decimal.
- Variables tipo String: Contienen cadenas de texto, o lo que es lo mismo, es un vector con varias variables del tipo Char.
- Variables del tipo Boolean: Solo pueden contener un 0 o un 1. El cero es considerado para muchos lenguajes como el literal "False", mientras que el 1 se considera "True".

2.2 Condicionantes

Los condicionantes son estructuras de código que indican que, para que cierta parte del programa se ejecute, deben cumplirse ciertas premisas; por ejemplo: que dos valores sean iguales, que un valor exista, que un valor sea mayor que otro... Estos condicionantes por lo general solo se ejecutan una vez a lo largo del programa. Los condicionantes más conocidos y empleados en programación son:

- **If**: Indica una condición para que se ejecute una parte del programa.
- **Else if**: Siempre va precedido de un "If" e indica una condición para que se ejecute una parte del programa siempre que no cumpla la condición del if previo y si se cumpla con la que el "else if" especifique.

- **Else**: Siempre precedido de “If” y en ocasiones de “Else If”. Indica que debe ejecutarse cuando no se cumplan las condiciones prévias.

2.3 Bucles

Los bucles son parientes cercanos de los condicionantes, pero ejecutan constantemente un código mientras se cumpla una determinada condición. Los más frecuentes son:

- **For**: Ejecuta un código mientras una variable se encuentre entre 2 determinados parámetros.
- **While**: Ejecuta un código mientras que se cumpla la condición que solicita.

Hay que decir que a pesar de que existan distintos tipos de bucles, ambos son capaces de realizar exactamente las mismas funciones. El empleo de uno u otro depende, por lo general, del gusto del programador.

2.4 Funciones

Las funciones se crearon para evitar tener que repetir constantemente fragmentos de código. Una función podría considerarse como una variable que encierra código dentro de si. Por lo tanto cuando accedemos a dicha variable (la función) en realidad lo que estamos es diciendo al programa que ejecute un determinado código predefinido anteriormente.

Todos los lenguajes de programación tienen algunos elementos de formación primitivos para la descripción de los datos y de los procesos o transformaciones aplicadas a estos datos (tal como la suma de dos números o la selección de un elemento que forma parte de una colección). Estos elementos primitivos son definidos por reglas sintácticas y semánticas que describen su estructura y significado respectivamente.

2.5 Sintaxis

A la forma visible de un lenguaje de programación se le conoce como sintaxis. La mayoría de los lenguajes de programación son puramente textuales, es decir, utilizan secuencias de texto que incluyen palabras, números y puntuación, de manera similar a los lenguajes naturales escritos. Por otra parte, hay algunos lenguajes de programación que son más gráficos en su naturaleza, utilizando relaciones visuales entre símbolos para especificar un programa.

La sintaxis de un lenguaje de programación describe las combinaciones posibles de los símbolos que forman un programa sintácticamente correcto. El significado que se le da a una combinación de símbolos es manejado por su

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

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

Con frecuencia se resaltan los elementos de la sintaxis con colores diferentes para facilitar su lectura. Este ejemplo está escrito en Python.

semántica (ya sea formal o como parte del código duro de la referencia de implementación). Dado que la mayoría de los lenguajes son textuales, este artículo trata de la sintaxis textual.

La sintaxis de los lenguajes de programación es definida generalmente utilizando una combinación de expresiones regulares (para la estructura léxica) y la Notación de Backus-Naur (para la estructura gramática). Este es un ejemplo de una gramática simple, tomada de Lisp:

expresión ::= átomo | lista átomo ::= número | símbolo número ::= [+-]? ['0'-'9']+ símbolo ::= ['A'-'Z'] ['a'-'z'].* lista ::= '(' expresión* ')'

Con esta gramática se especifica lo siguiente:

- una *expresión* puede ser un *átomo* o una *lista*;
- un *átomo* puede ser un *número* o un *símbolo*;
- un *número* es una secuencia continua de uno o más dígitos decimales, precedido opcionalmente por un signo más o un signo menos;
- un *símbolo* es una letra seguida de cero o más caracteres (excluyendo espacios); y
- una *lista* es un par de paréntesis que abren y cierran, con cero o más expresiones en medio.

Algunos ejemplos de secuencias bien formadas de acuerdo a esta gramática:

'12345', '()', '(a b c232 (1))'

No todos los programas sintácticamente correctos son semánticamente correctos. Muchos programas sintácticamente correctos tienen inconsistencias con las reglas del lenguaje; y pueden (dependiendo de la especificación del lenguaje y la solidez de la implementación) resultar en un error de traducción o ejecución. En algunos casos, tales

programas pueden exhibir un comportamiento indefinido. Además, incluso cuando un programa está bien definido dentro de un lenguaje, todavía puede tener un significado que no es el que la persona que lo escribió estaba tratando de construir.

Usando el lenguaje natural, por ejemplo, puede no ser posible asignarle significado a una oración gramaticalmente válida o la oración puede ser falsa:

- "Las ideas verdes y descoloridas duermen furiosamente" es una oración bien formada gramaticalmente pero no tiene significado comúnmente aceptado.
- "Juan es un soltero casado" también está bien formada gramaticalmente pero expresa un significado que no puede ser verdadero.

El siguiente fragmento en el lenguaje C es sintácticamente correcto, pero ejecuta una operación que no está definida semánticamente (dado que p es un apuntador nulo, las operaciones p->real y p->im no tienen ningún significado):

complex *p = NULL; complex abs_p = sqrt (p->real * p->real + p->im * p->im);

Si la declaración de tipo de la primera línea fuera omitida, el programa dispararía un error de compilación, pues la variable "p" no estaría definida. Pero el programa sería sintácticamente correcto todavía, dado que las declaraciones de tipo proveen información semántica solamente.

La gramática necesaria para especificar un lenguaje de programación puede ser clasificada por su posición en la Jerarquía de Chomsky. La sintaxis de la mayoría de los lenguajes de programación puede ser especificada utilizando una gramática Tipo-2, es decir, son gramáticas libres de contexto. Algunos lenguajes, incluyendo a Perl y a Lisp, contienen construcciones que permiten la ejecución durante la fase de análisis. Los lenguajes que permiten construcciones que permiten al programador alterar el comportamiento de un analizador hacen del análisis de la sintaxis un problema sin decisión única, y generalmente oscurecen la separación entre análisis y ejecución. En contraste con el sistema de macros de Lisp y los bloques BEGIN de Perl, que pueden tener cálculos generales, las macros de C son meros reemplazos de cadenas, y no requieren ejecución de código.

2.6 Semántica estática

La semántica estática define las restricciones sobre la estructura de los textos válidos que resulta imposible o muy difícil expresar mediante formalismos sintácticos estándar. Para los lenguajes compilados, la semántica estática básicamente incluye las reglas semánticas que se pueden verificar en el momento de compilar. Por ejemplo el chequeo de que cada identificador sea declarado antes de ser usado (en lenguajes que requieren tales declaraciones) o que las etiquetas en cada brazo de una estructura *case* sean distintas. Muchas restricciones importantes de este tipo, como la validación de que los identificadores sean usados en los contextos apropiados (por ejemplo no sumar un entero al nombre de una función), o que las llamadas a subrutinas tengan el número y tipo de parámetros adecuado, puede ser implementadas definiéndolas como reglas en una lógica conocida como sistema de tipos. Otras formas de análisis estáticos, como los análisis de flujo de datos, también pueden ser parte de la semántica estática. Otros lenguajes de programación como Java y C# tienen un análisis definido de asignaciones, una forma de análisis de flujo de datos, como parte de su semántica estática.

2.7 Sistema de tipos

Un sistema de tipos define la manera en la cual un lenguaje de programación clasifica los valores y expresiones en *tipos*, cómo pueden ser manipulados dichos tipos y cómo interactúan. El objetivo de un sistema de tipos es verificar y normalmente poner en vigor un cierto nivel de exactitud en programas escritos en el lenguaje en cuestión, detectando ciertas operaciones inválidas. Cualquier sistema de tipos decidible tiene sus ventajas y desventajas: mientras por un lado rechaza muchos programas incorrectos, también prohíbe algunos programas correctos aunque poco comunes. Para poder minimizar esta desventaja, algunos lenguajes incluyen *lagunas de tipos*, conversiones explícitas no checadas que pueden ser usadas por el programador para permitir explícitamente una operación normalmente no permitida entre diferentes tipos. En la mayoría de los lenguajes con tipos, el sistema de tipos es usado solamente para checar los tipos de los programas, pero varios lenguajes, generalmente funcionales, llevan a cabo lo que se conoce como inferencia de tipos, que le quita al programador la tarea de especificar los tipos. Al diseño y estudio formal de los sistemas de tipos se le conoce como *teoría de tipos*.

2.7.1 Lenguajes tipados versus lenguajes no tipados

Se dice que un lenguaje tiene *tipos* si la especificación de cada operación define tipos de datos para los cuales la operación es aplicable, con la implicación de que no es aplicable a otros tipos. Por ejemplo, "este texto entre comillas" es una cadena. En la mayoría de los lenguajes de programación, dividir un número por una cadena no tiene ningún significado. Por tanto, la mayoría de los lenguajes de programación modernos rechazaran cualquier intento de ejecutar dicha operación por parte de algún programa. En algunos lenguajes, estas operaciones sin significado son detectadas cuando el programa es compilado (validación de tipos "estática") y son rechazadas por el compilador, mientras en otros son detectadas cuando el programa es ejecutado (validación de tipos "dinámica") y se genera

una excepción en tiempo de ejecución.

Un caso especial de lenguajes de tipo son los lenguajes de *tipo sencillo*. Estos son con frecuencia lenguajes de marcado o de scripts, como REXX o SGML, y solamente cuentan con un tipo de datos; comúnmente cadenas de caracteres que luego son usadas tanto para datos numéricos como simbólicos.

En contraste, un lenguaje *sin tipos*, como la mayoría de los lenguajes ensambladores, permiten que cualquier operación se aplique a cualquier dato, que por lo general se consideran secuencias de bits de varias longitudes. Lenguajes de alto nivel *sin datos* incluyen BCPL y algunas variedades de Forth.

En la práctica, aunque pocos lenguajes son considerados con tipo desde el punto de vista de la teoría de tipos (es decir, que verifican o rechazan *todas* las operaciones), la mayoría de los lenguajes modernos ofrecen algún grado de manejo de tipos. Si bien muchos lenguajes de producción proveen medios para brincarse o subvertir el sistema de tipos.

2.7.2 Tipos estáticos versus tipos dinámicos

En lenguajes con *tipos estáticos* se determina el tipo de todas las expresiones antes de la ejecución del programa (típicamente al compilar). Por ejemplo, 1 y (2+2) son expresiones enteras; no pueden ser pasadas a una función que espera una cadena, ni pueden guardarse en una variable que está definida como fecha.

Los lenguajes con tipos estáticos pueden manejar tipos *explícitos* o tipos *inferidos*. En el primer caso, el programador debe escribir los tipos en determinadas posiciones textuales. En el segundo caso, el compilador *infiere* los tipos de las expresiones y las declaraciones de acuerdo al contexto. La mayoría de los lenguajes populares con tipos estáticos, tales como C++, C# y Java, manejan tipos explícitos. Inferencia total de los tipos suele asociarse con lenguajes menos populares, tales como Haskell y ML. Sin embargo, muchos lenguajes de tipos explícitos permiten inferencias parciales de tipo; tanto Java y C#, por ejemplo, infieren tipos en un número limitado de casos.

Los lenguajes con tipos dinámicos determinan la validez de los tipos involucrados en las operaciones durante la ejecución del programa. En otras palabras, los tipos están asociados con *valores en ejecución* en lugar de *expresiones textuales*. Como en el caso de lenguajes con tipos inferidos, los lenguajes con tipos dinámicos no requieren que el programador escriba los tipos de las expresiones. Entre otras cosas, esto permite que una misma variable se pueda asociar con valores de tipos distintos en diferentes momentos de la ejecución de un programa. Sin embargo, los errores de tipo no pueden ser detectados automáticamente hasta que se ejecuta el código, dificultando la depuración de los programas, no obstante, en lenguajes con tipos dinámicos se suele dejar de lado la depuración en favor de técnicas de desarrollo como por ejemplo BDD y TDD. Ruby, Lisp, JavaScript y Python son lenguajes con tipos dinámicos.

2.7.3 Tipos débiles y tipos fuertes

Los lenguajes *débilmente tipados* permiten que un valor de un tipo pueda ser tratado como de otro tipo, por ejemplo una cadena puede ser operada como un número. Esto puede ser útil a veces, pero también puede permitir ciertos tipos de fallas que no pueden ser detectadas durante la compilación o a veces ni siquiera durante la ejecución.

Los lenguajes *fuertemente tipados* evitan que pase lo anterior. Cualquier intento de llevar a cabo una operación sobre el tipo equivocado dispara un error. A los lenguajes con tipos fuertes se les suele llamar *de tipos seguros*.

Lenguajes con tipos débiles como Perl y JavaScript permiten un gran número de conversiones de tipo implícitas. Por ejemplo en JavaScript la expresión 2 * x convierte implícitamente x a un número, y esta conversión es exitosa inclusive cuando x es null, undefined, un Array o una cadena de letras. Estas conversiones implícitas son útiles con frecuencia, pero también pueden ocultar errores de programación.

Las características de *estáticos* y *fuertes* son ahora generalmente consideradas conceptos ortogonales, pero su trato en diferentes textos varia. Algunos utilizan el término *de tipos fuertes* para referirse a *tipos fuertemente estáticos* o, para aumentar la confusión, simplemente como equivalencia de *tipos estáticos*. De tal manera que C ha sido llamado tanto lenguaje de tipos fuertes como lenguaje de tipos estáticos débiles.

3 Implementación

```
/**
 * Simple HelloButton() method.
 * @version 1.0
 * @author john doe <doe.j@example.com>
 */
HelloButton()
{
  JButton hello = new JButton( "Hello, wor
  hello.addActionListener( new HelloBtnList

  // use the JFrame type until support for t
  // new component is finished
  JFrame frame = new JFrame( "Hello Button"
  Container pane = frame.getContentPane();
  pane.add( hello );
  frame.pack();
  frame.show();           // display the fra
}
```

Código fuente de un programa escrito en el lenguaje de programación Java.

La implementación de un lenguaje es la que provee una manera de que se ejecute un programa para una determinada combinación de software y hardware. Existen básicamente dos maneras de implementar un lenguaje: compilación e interpretación.

- Compilación: es el proceso que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz interpretar. Los programas traductores que pueden realizar esta operación se llaman compiladores. Éstos, como los programas ensambladores avanzados, pueden generar muchas líneas de código de máquina por cada proposición del programa fuente.
- Interpretación: es una asignación de significados a las fórmulas bien formadas de un lenguaje formal. Como los lenguajes formales pueden definirse en términos puramente sintácticos, sus fórmulas bien formadas pueden no ser más que cadenas de símbolos sin ningún significado. Una interpretación otorga significado a esas fórmulas.

Se puede también utilizar una alternativa para traducir lenguajes de alto nivel. En lugar de traducir el programa fuente y grabar en forma permanente el código objeto que se produce durante la compilación para utilizarlo en una ejecución futura, el programador sólo carga el programa fuente en la computadora junto con los datos que se van a procesar. A continuación, un programa intérprete, almacenado en el sistema operativo del disco, o incluido de manera permanente dentro de la máquina, convierte cada proposición del programa fuente en lenguaje de máquina conforme vaya siendo necesario durante el procesamiento de los datos. El código objeto no se graba para utilizarlo posteriormente.

La siguiente vez que se utilice una instrucción, se la deberá interpretar otra vez y traducir a lenguaje máquina. Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo o bucle, cada instrucción del bucle tendrá que volver a ser interpretada en cada ejecución repetida del ciclo, lo cual hace que el programa sea más lento en tiempo de ejecución (porque se va revisando el código en tiempo de ejecución) pero más rápido en tiempo de diseño (porque no se tiene que estar compilando a cada momento el código completo). El intérprete elimina la necesidad de realizar una compilación después de cada modificación del programa cuando se quiere agregar funciones o corregir errores; pero es obvio que un programa objeto compilado con antelación deberá ejecutarse con mucha mayor rapidez que uno que se debe interpretar a cada paso durante una ejecución del código.

La mayoría de lenguajes de alto nivel permiten la programación multipropósito, aunque muchos de ellos fueron diseñados para permitir programación dedicada, como lo fue el Pascal con las matemáticas en su comienzo. También se han implementado lenguajes educativos infantiles como Logo mediante una serie de simples instrucciones. En la actualidad son muy populares algunos lenguajes especialmente indicados para aplicaciones web, como Perl, PHP, Ruby, Python o JavaScript.

4 Técnica

Libros sobre diversos lenguajes de programación.

Para escribir programas que proporcionen los mejores resultados, cabe tener en cuenta una serie de detalles.

- *Corrección*. Un programa es correcto si hace lo que debe hacer tal y como se estableció en las fases previas a su desarrollo. Para determinar si un programa hace lo que debe, es muy importante especificar claramente qué debe hacer el programa antes de desarrollarlo y, una vez acabado, compararlo con lo que realmente hace.

- *Claridad*. Es muy importante que el programa sea lo más claro y legible posible, para facilitar así su desarrollo y posterior mantenimiento. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo en la edición; de esta forma se ve facilitado el trabajo del programador, tanto en la fase de creación como en las fases posteriores de corrección de errores, ampliaciones, modificaciones, etc. Fases que pueden ser realizadas incluso por otro programador, con lo cual la claridad es aún más necesaria para que otros programadores puedan continuar el trabajo fácilmente. Algunos programadores llegan incluso a utilizar Arte ASCII para delimitar secciones de código. Otros, por diversión o para impedir un análisis cómodo a otros programadores, recurren al uso de código ofuscado.

- *Eficiencia*. Se trata de que el programa, además de realizar aquello para lo que fue creado (es decir, que sea correcto), lo haga gestionando de la mejor forma posible los recursos que utiliza. Normalmente, al hablar de eficiencia de un programa, se suele hacer

referencia al tiempo que tarda en realizar la tarea para la que ha sido creado y a la cantidad de memoria que necesita, pero hay otros recursos que también pueden ser de consideración al obtener la eficiencia de un programa, dependiendo de su naturaleza (espacio en disco que utiliza, tráfico de red que genera, etc.).

- *Portabilidad*. Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea hardware o software, diferente a aquella en la que se elaboró. La portabilidad es una característica muy deseable para un programa, ya que permite, por ejemplo, a un programa que se ha desarrollado para sistemas GNU/Linux ejecutarse también en la familia de sistemas operativos Windows. Esto permite que el programa pueda llegar a más usuarios más fácilmente.

4.1 Paradigmas

Los programas se pueden clasificar por el paradigma del lenguaje que se use para producirlos. Los principales paradigmas son: imperativos, declarativos y orientación a objetos.

Los programas que usan un lenguaje imperativo especifican un algoritmo, usan declaraciones, expresiones y sentencias.[3] Una declaración asocia un nombre de variable con un tipo de dato, por ejemplo: var x: integer;. Una expresión contiene un valor, por ejemplo: 2 + 2 contiene el valor 4. Finalmente, una sentencia debe asignar una expresión a una variable o usar el valor de una variable para alterar el flujo de un programa, por ejemplo: x := 2 + 2; if x == 4 then haz_algo();. Una crítica común en los lenguajes imperativos es el efecto de las sentencias de asignación sobre una clase de variables llamadas “no locales”.[4]

Los programas que usan un lenguaje declarativo especifican las propiedades que la salida debe conocer y no especifica cualquier detalle de implementación. Dos amplias categorías de lenguajes declarativos son los lenguajes funcionales y los lenguajes lógicos. Los lenguajes funcionales no permiten asignaciones de variables no locales, así, se hacen más fácil, por ejemplo, programas como funciones matemáticas.[4] El principio detrás de los lenguajes lógicos es definir el problema que se quiere resolver (el objetivo) y dejar los detalles de la solución al sistema.[5] El objetivo es definido dando una lista de sub-objetivos. Cada sub-objetivo también se define dando una lista de sus sub-objetivos, etc. Si al tratar de buscar una solución, una ruta de sub-objetivos falla, entonces tal sub-objetivo se descarta y sistemáticamente se prueba otra ruta.

La forma en la cual se programa puede ser por medio de texto o de forma visual. En la programación visual los elementos son manipulados gráficamente en vez de especificarse por medio de texto.

5 Véase también

- Anexo:Lenguajes de programación
- Programación estructurada
- Programación modular
- Programación orientada a objetos
- Programación imperativa
- Programación declarativa
- paradigma de programación
- Lenguajes esotéricos

6 Referencias

[1] [Mark] Comprueba el valor del lenlaceautor= (ayuda) (2010). O'Reilly Media, Inc., ed. «Learning Python, Fourth Edition» (libro). O'Reilly. Consultado el 11 de febrero de 2010. lautor= y lapellido= redundantes (ayuda)

[2] http://www.softwarepreservation.org/projects/FORTRAN/index.html#By_FORTRAN_project_members

[3] Wilson, Leslie B. (1993). *Comparative Programming Languages, Second Edition*. Addison-Wesley. p. 75. ISBN 0-201-56885-3. (en inglés).

[4] Wilson, Leslie B. (1993). *Comparative Programming Languages, Second Edition*. Addison-Wesley. p. 213. ISBN 0-201-56885-3. (en inglés).

[5] Wilson, Leslie B. (1993). *Comparative Programming Languages, Second Edition*. Addison-Wesley. p. 244. ISBN 0-201-56885-3. (en inglés).

7 Enlaces externos

- Wikimedia Commons alberga contenido multimedia sobre **Lenguaje de programación**. Commons
- Wikiversidad alberga proyectos de aprendizaje sobre **Lenguaje de programación**.Wikiversidad

Wikilibros

- Wikilibros alberga un libro o manual sobre **Fundamentos de programación**.
- Árbol genealógico de los lenguajes de programación (en inglés)
- Lista de lenguajes de programación (en inglés)
- Lenguajes clasificados por paradigmas de programación: definiciones, ventajas y desventajas.

8 Text and image sources, contributors, and licenses

8.1 Text

- **Lenguaje de programación** *Fuente:* http://es.wikipedia.org/wiki/Lenguaje%20de%20programaci%C3%B3n?oldid=79621562 *Colaboradores:* AstroNomo, Carlita, Nnss, Joseaperez, Moriel, Sauron, JorgeGG, ManuelGR, Julie, Angus, Rumpelstiltskin, Achury, Sanbec, Zwobot, Javier Carro, Scott MacLean, MiguelRdz, Dodo, Triku, Ascánder, Avm, Rsg, HHM, Cookie, Tostadora, Elwikipedista, Danakil, Murphy era un optimista, Jsanchezes, Jarfil, Elproferoman, Melocoton, Kalcetin, Porao, Trylks, Almorca, Suruena, Balderai, Dat, Niqueco, ZackBsAs, Ilario, LeonardoRob0t, Digigalos, Zeioth, Soulreaper, AlfonsoERomero, Airunp, JMPerez, Edub, Taichi, Rembiapo pohyiete (bot), Genba, Magister Mathematicae, Aadrover, Orgullobot, RobotQuistnix, Gcsantiago, Platonides, Unf, LarA, Alhen, Superzerocool, Chobot, Yrbot, BOT-Superzerocool, Oscar ., Martincarr, Vitamine, Cesarsorm, Wiki-Bot, Icvav, GermanX, AalvaradoH, Indu, KnightRider, Jesuja, Tigerfenix, Santiperez, PabloBD, Banfield, Otermin, Nowadays, Er Komandante, Spc, Tomatejc, Balix, Rbonvall, Jorgechp, Juanjo64, BOTpolicia, Qwertyytrewqqwerty, Chfiguer, CEM-bot, Jorgelrm, Krli2s, Spazer, Alex15090, Xemuj, Ignacio Icke, Retama, Baiji, Ezequiel3E, Osepu, Roberpl, JoRgE-1987, Eamezaga, Antur, Mr. Moonlight, FrancoGG, Fsd141, Juank8041, Alvaro qc, Srengel, Cansado, Mahadeva, Rata blanca, Diosa, Escarbot, RoyFocker, Eduiba, IrwinSantos, Will vm, Tintinando, Botones, Cratón, Isha, Arcibel, Mpeinadopa, JAnDbot, Thormaster, Jugones55, BelegDraug, Diego.souto, Kved, TitoPeru, Mansoncc, BetBot, Xavigivax, Danielantonionunez, Bot-Schafter, Elisardojm, Humberto, Netito777, Amanuense, Bedwyr, Idioma-bot, Pólux, Sebado, Bucephala, AlnoktaBOT, Dusan, Cinevoro, VolkovBot, Carola-zzz, Snakeyes, Technopat, Queninosta, Aliamondano, Pejeyo, Matdrodes, Elabra sanchez, Synthebot, House, DJ Nietzsche, Gorpik, Lucien leGrey, AlleborgoBot, Gmarinp, IIM 78, Muro Bot, Edmenb, J.M.Domingo, Komputisto, Bucho, SieBot, Mushii, Danielba894, Carlos Zeas, Cobalttempest, Rigenea, Hompis, Drinibot, Bigsus-bot, IsmaelLuceno, Mel 23, Manwë, Correogsk, Greek, Joseromerogc, BuenaGente, Mafores, Cam367, Tirithel, Jarisleif, Javierito92, HUB, FCPB, Eduardosalg, Edgarchan, Leonpolanco, Pan con queso, Alecs.bot, LordT, Descansatore, Petruss, Poco a poco, Juan Mayordomo, Darkicebot, Rœge, MADAFACK, Frei sein, Raulshc, Açipni-Lovrij, Majin boo, Camilo, UA31, Inakivk, Esterdelakpaz, CharlesKnight, AVBOT, Swatnio, DayL6, David0811, J.delanoy, MarcoAurelio, JyQ, Ezarate, Diegusjaimes, MelancholieBot, Linfocito B, Sebasweee, Arjuno3, Andreasmperu, Luckas-bot, MystBot, Bob A, Roinpa, Jotterbot, Sappler, Akhran, Vic Fede, Barteik, Billinghurst, Vandal Crusher, Julio Cardmat, XZeroBot, ArthurBot, Argentinoo, Jefrcast, SuperBraulio13, Ortisa, Manuelt15, Xqbot, Jkbw, SassoBot, Dreitmen, Ricardogpn, Igna, Torrente, Adryitan, Botarel, MauritsBot, Googolplanck, Hprmedina, TobeBot, Execoot, Palita1880, Halfdrag, Vubo, YSCO, PatruBOT, Ganímedes, Angelito7, Mr.Ajedrez, Jose1080i, Humbefa, Tarawa1943, Jorge c2010, GrouchoBot, Cesarintel, Miss Manzana, Jose1100000, Edslov, EmausBot, Savh, AVIADOR, Ingenieros instructivos, HRoestBot, Allforrous, Sergio Andres Segovia, Chope777, Rubpe19, Eroyuela, ChuispastonBot, MadriCR, Waka Waka, Cordwainer, Lcsrns, Antonorsi, Valthern, Edc.Edc, Ismaell, Jicapter, Sebrev, MetroBot, Sansgumen, Mascorria, Teclis jma, Gusama Romero, Nernix1, Acratta, Johnbot, Harpagornis, LlamaAl, Helmy oved, Erandly, Flashlack, Cyrax, Napier, Ralgisbot, Syum90, Fabrizotus, BLACK M0NST3R, Juanitoalcachofados, Addbot, Shinigamisajayin, Balles2601, Killtas, Patoogalvan, Davidpaisa04, Equiz yolo swaw, Matiia, Egis57, Geekisthebest1 y Anónimos: 784

8.2 Images

- **Archivo:Classes_and_Methods.png** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/d/d0/Classes_and_Methods.png *Licencia:* CC BY-SA 3.0 *Colaboradores:* ? *Artista original:* ?
- **Archivo:CodeCmmt002.svg** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/7/75/CodeCmmt002.svg *Licencia:* CC BY 2.5 *Colaboradores:* Originally from en.wikipedia; description page is/was here. *Artista original:* Original uploader was Dreftymac at en.wikipedia
- **Archivo:Commons-logo.svg** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/4/4a/Commons-logo.svg *Licencia:* Public domain *Colaboradores:* This version created by Pumbaa, using a proper partial circle and SVG geometry features. (Former versions used to be slightly warped.) *Artista original:* SVG version was created by User:Grunt and cleaned up by 3247, based on the earlier PNG version, created by Reidab.
- **Archivo:FortranCardPROJ039.agr.jpg** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/5/58/FortranCardPROJ039.agr.jpg *Licencia:* CC BY-SA 2.5 *Colaboradores:* I took this picture of an artifact in my possession. The card was created in the late 1960s or early 1970s and has no copyright notice. *Artista original:* Arnold Reinhold
- **Archivo:PET-basic.png** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/0/0b/PET-basic.png *Licencia:* Public domain *Colaboradores:* Trabajo propio *Artista original:* Rafax
- **Archivo:Programming_language_textbooks.jpg** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/a/a0/Programming_language_textbooks.jpg *Licencia:* Public domain *Colaboradores:* Trabajo propio *Artista original:* User:K.lee
- **Archivo:Python_add5_syntax.svg** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/e/e1/Python_add5_syntax.svg *Licencia:* Copyrighted free use *Colaboradores:* http://en.wikipedia.org/wiki/Image:Python_add5_syntax.png *Artista original:* Xander89
- **Archivo:Translation_arrow.svg** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/2/2a/Translation_arrow.svg *Licencia:* CC-BY-SA-3.0 *Colaboradores:* Este imagen vectorial fue creado con Inkscape *Artista original:* Jesse Burgheimer
- **Archivo:Wikibooks-logo.svg** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/f/fa/Wikibooks-logo.svg *Licencia:* CC BY-SA 3.0 *Colaboradores:* Trabajo propio *Artista original:* User:Bastique, User:Ramac et al.
- **Archivo:Wikiversity-logo-Snorky.svg** *Fuente:* http://upload.wikimedia.org/wikipedia/commons/1/1b/Wikiversity-logo-en.svg *Licencia:* CC BY-SA 3.0 *Colaboradores:* Trabajo propio *Artista original:* Snorky

8.3 Content license

- Creative Commons Attribution-Share Alike 3.0