Fascículo 4

Programación en diversos lenguajes

Cuaderno 2: Lenguajes de programación: sus componentes

Inicios de la programación y evolución de los lenguajes

Explicamos en el primer cuaderno que la programación es la disciplina que se orienta al proceso de creación de un programa de computadora; es tiempo de conocer más la historia de su evolución.

La primera programadora de computadora reconocida fue Ada Lovelace (1815-1852), hija de Anabella Milbanke Byron y el poeta Lord Byron. A principios del siglo XIX conoció a Charles Babbage, un inventor inglés y profesor matemático de la universidad de Cambridge, que diseñó –pero nunca construyó– la máquina analítica para ejecutar programas de tabulación, por lo que se lo considera como el "padre" de la computación.

Fue Ada Lovelace quien predijo muchas de las teorías actuales al traducir y ampliar una descripción de la máquina analítica de Babbage, uno de los antecedentes más directos de lo que conocemos como computadora. Como la máquina no llegó nunca a construirse, los programas de Ada lógicamente tampoco llegaron a ejecutarse, pero sí suponen un punto de partida de la programación.

El trabajo que Ada realizó le hizo ganarse el título de primera programadora de computadoras del mundo. El nombre del **lenguaje de programación Ada**, utilizado principalmente en aeronáutica, fue escogido en su homenaje.

Casi cien años después, a finales de 1954, para evitar las dificultades de programación de las calculadoras de su época, el informático estadounidense John Backus, se encargó de la dirección de un proyecto de investigación en IBM para el desarrollo de un lenguaje de programación más cercano a la notación matemática normal.

De ese proyecto surgió el **lenguaje Fortran**, el primero de los lenguajes de programación de alto nivel, que tuvo un gran impacto, incluso comercial, en la emergente comunidad informática.

En 1960, se creó **COBOL** (*COmmon Business -Oriented Language,* o lenguaje común orientado a negocios), uno de los lenguajes usados aún hoy en informática de gestión. Respondió al objetivo de contar con un **lenguaje de programación universal** que pudiera ser usado en cualquier computadora –ya que en los años 1960 existían numerosos modelos incompatibles entre sí– y que estuviera orientado principalmente a los negocios, es decir, a la llamada informática de gestión.

Pero aún en la década de 1960 las computadoras eran máquinas sumamente caras que se utilizaban únicamente para propósitos especiales, y ejecutaban una sola tarea a la vez. Sin embargo, durante ese período, los precios comenzaron a bajar al punto de que incluso las pequeñas empresas podían comprarlas. 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 (*Beginners All-purpose Symbolic Instruction Codeen*, o código de instrucciones simbólicas de propósito general para principiantes) y otras versiones introducidas en las computadoras que se utilizaron a partir de la década de 1980.

> Habrán observado que para hablar de la historia de la programación tenemos que hablar de la historia de los lenguajes de programación. Estos lenguajes y muchos otros que no se mencionan aquí no son totalmente independientes entre sí, sino que unos influyeron en el diseño de los otros, por lo que es difícil establecer una jerarquía histórica.

Para "hablarle" a una computadora es necesario utilizar un lenguaje en particular: el único lenguaje que una computadora entiende se denomina binario y tiene muchos dialectos. Esto demuestra por qué un programa escrito para una iMac a veces no funciona en una PC (*Personal Computer,* o computadora personal) y viceversa.

Desafortunadamente el lenguaje binario es muy difícil de leer y escribir, por lo cual se debe utilizar un **lenguaje intermedio** que después será traducido a binario.

Lo que traduce nuestro lenguaje intermediario a binario se denomina **intérprete**.

De la misma manera que es necesario disponer de un intérprete para traducir del inglés al ruso, será necesario disponer de un intérprete también para traducir las órdenes de, por ejemplo, Python a binario.

Del código máquina a los lenguajes de alto nivel

Los primeros programadores tenían que ingresar los códigos binarios. Esta acción se conoce como programación en **código máquina**, y es increíblemente compleja.

No pasó mucho tiempo hasta que se pudo desarrollar un **traductor** que simplemente convertía palabras en inglés a su equivalente en código binario.

De esta manera, en vez de tener que recordar que el código 001273 05 04 significaba "sumar 5 + 4", los programadores podían escribir entonces "**ADD"** (sumar en inglés) **5 4**.

Esta mejora hizo que la programación fuera más sencilla y que surgieran velozmente los primeros lenguajes de programación y las distintas versiones para cada tipo de computadora.

El desarrollo de estas versiones se conoce como **lenguajes ensambladores**, y aún se utilizan para algunas tareas de programación muy específicas. En otras palabras, el ensamblador es un tipo de **lenguaje de bajo nivel** utilizado para escribir programas informáticos específicos para cada arquitectura de computadoras.

Originalmente este sistema era muy primitivo, pues le decía a la computadora lo que tenía que hacer en el nivel de hardware. Lograr un objetivo sencillo era todavía bastante difícil e implicaba un gran esfuerzo de programación.

> Los **lenguajes de alto nivel** son actualmente los más utilizados en programación. Aunque no son fundamentalmente declarativos, estos lenguajes permiten que los algoritmos se expresen en un nivel y estilo de escritura fácilmente legible y comprensible por otros programadores.

Además, los lenguajes de alto nivel tienen normalmente la característica de **transportabilidad**. Es decir, están implementadas sobre varias máquinas de forma que un programa puede ser fácilmente transportado o transferido de una máquina a otra sin una revisión sustancial. En ese sentido se llaman "**independientes de la máquina**".

Algunos ejemplos de estos lenguajes de alto nivel son:

PASCAL, APL y FORTRAN	Para aplicaciones científicas.
COBOL	Para aplicaciones de procesamiento de datos.
SNOBOL	Para aplicaciones de procesamiento de textos.
LISP y PROLOG	Para aplicaciones de inteligencia artificial.
C y ADA	Para aplicaciones de programación de sistemas.
PL/I	Para aplicaciones de propósitos generales.

Los **lenguajes declarativos** son los más parecidos al castellano o inglés en su potencia expresiva y funcionalidad: están en un nivel más alto respecto de los otros. Son fundamentalmente **lenguajes de órdenes**, dominados por sentencias que expresan "lo que hay que hacer".

Ejemplos de estos lenguajes son los lenguajes estadísticos como SAS y SPSS y los lenguajes de búsqueda en base de datos, como NATURAL e IMS. Se desarrollaron con la idea de que los programadores pudieran asimilar más rápidamente el lenguaje y usarlo en su trabajo.

Sintaxis, semántica y gramática del lenguaje de programación

Se conoce como **sintaxis** a la parte visible de un lenguaje de programación. Se define como el **conjunto de reglas** que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

La mayoría de los lenguajes de programación son puramente textuales, es decir, utilizan **secuencias de texto** que incluyen palabras, números y signos de pun-

tuación, de manera similar a los lenguajes naturales escritos. Por otra parte, hay algunos lenguajes de programación que son más gráficos en su naturaleza y utilizan relaciones visuales entre símbolos para especificar un programa.

La sintaxis de un lenguaje de programación describe además 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 semántica.**

La sintaxis de los lenguajes de programación se define mediante la utilización de una combinación de expresiones. Veamos como caso esta gramática simple, tomada de Lisp, una familia de lenguajes de programación de computadora de tipo funcional con una larga historia y una sintaxis desarrollada completamente entre paréntesis.

Desarrollado originalmente en 1958, es el segundo más viejo lenguaje de programación de alto nivel (el primero es el Fortran).

```
expresión ::= átomo | lista
atomo     ::= 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).
- Una lista es un par de paréntesis que abren y cierran, con cero o más expresiones en medio.

Las **reglas** que determinan el significado de los programas constituyen la **semántica** de los lenguajes de programación.

Es importante saber que no todos los programas **sintácticamente correctos** son **semánticamente correctos**. Muchos programas sintácticamente correctos tienen inconsistencias respecto de 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:

Los ideales verdes y descoloridos duermen estrepitosamente, es una oración bien formada gramaticalmente, pero no tiene significado comúnmente aceptado.

Manuel es un soltero casado también está bien formada gramaticalment, pero expresa un significado inválido, contradictorio.

> Es decir que un lenguaje de programación consta de un conjunto de símbolos y un conjunto de reglas válidas para componerlos, para que conformen un mensaje con significado para la computadora.

En **síntesis**, los lenguajes de programación constan de:

- Un conjunto finito de símbolos, a partir del cual se define el léxico o vocabulario del lenguaje.
- Un conjunto finito de reglas, la gramática del lenguaje, para la construcción de las sentencias correctas del lenguaje (sintaxis).
- Semántica, que asocia un significado –la acción que debe llevarse a cabo– a cada posible construcción del lenguaje.

Sistema de tipos

El lenguaje de programación debe además clasificar los valores y expresiones en **tipos**, los cuales conforman un sistema cuyo objetivo **es verificar el funcionamiento del programa y detectar operaciones inválidas**.

Un sistema de tipos dota a los lenguajes de la capacidad de restringir los datos que pueden ser asignados a las variables. Esto permite una cierta potencia a la hora de detectar errores y mejora la comprensión del código.

Cualquier sistema de tipos tiene sus ventajas y desventajas: mientras que por un lado rechaza muchos programas incorrectos, también prohíbe algunos programas correctos que por alguna razón le resulten desconocidos.

Para poder minimizar esta desventaja, algunos lenguajes incluyen **lagunas de tipos**, que son conversiones explícitas no chequeadas que pueden ser usadas por el programador para permitir explícitamente una operación normalmente no permitida entre diferentes tipos.

Para sintetizar, el sistema de tipos común realiza las siguientes **funciones**:

- Establece un marco de trabajo que ayuda a permitir la integración entre lenguajes, la seguridad de tipos y la ejecución de código con alto rendimiento.

- Proporciona un modelo orientado a objetos que admite la implementación completa de muchos lenguajes de programación.

- Define reglas que deben seguir los lenguajes, lo que ayuda a garantizar que los objetos escritos en distintos lenguajes puedan interactuar unos con otros.

Al diseño y estudio formal de los sistemas de tipos se le conoce como **teoría de tipos**.

Si les interesa profundizar sobre esta teoría pueden consultar este material disponible en internet.

¿Les parece interesante –aunque seguramente algo complejo– lo que vieron hasta aquí? Tranquilos, no son los únicos… Hace no mucho tiempo, a finales de 1960, Edsger Dijkstra, un científico de la computación de origen holandés, trató de simplificar los pasos de la programación. Desarrolló el concepto de la **programación estructurada** y definió que todos los programas pueden estructurarse en los siguientes pasos:

- **Secuencias de instrucciones**
  Una estructura de programa es secuencial si se ejecuta una tras otra a modo de secuencia, es decir que una instrucción no se ejecuta hasta que finaliza la anterior.

- **Instrucción condicional**
  La estructura selectiva permite la realización de una instrucción u otra según un criterio: solo una de estas instrucciones se ejecutará.

- **Iteración (bucle de instrucciones)**
  Un bucle iterativo o iteración de una secuencia de instrucciones hace que se repitan mientras se cumpla una condición: en un principio el número de iteraciones no tiene por qué estar determinado.

Si bien con esta teoría los programas pueden ser más fáciles de entender, hoy en día las aplicaciones informáticas son mucho más ambiciosas que las necesidades de programación existentes en los años 60 –principalmente debido a las aplicaciones gráficas– por lo que las técnicas de programación estructurada no son suficientes.

Ello ha llevado al desarrollo de nuevas técnicas, tales como la **programación orientada a objetos** y el desarrollo de **entornos de programación** que facilitan la programación de grandes aplicaciones.

Gradualmente los expertos en computación desarrollaron lenguajes de alto nivel para facilitar el trabajo de los programadores. Esto fue también el resultado de una demanda de los usuarios que reclamaban tareas más complejas y procesos más potentes para sus computadoras. Esta exigencia de los usuarios hacia los programadores continúa en la actualidad y cada vez son más los especialistas dedicados a desarrollar y potenciar nuevos lenguajes. Esto vuelve muy interesante a la programación como disciplina.

En el siguiente cuaderno analizaremos con profundidad algunos de los perfiles profesionales del programador.

Fuentes

- www.wikipedia.org

Autora: María Lorena Suárez
Coordinación editorial: Mara Mobilia