HISTORIA DE LA COMPUTACIÓN

Uno de los primeros dispositivos mecánicos para contar fue el ábaco, cuya historia se remonta a las antiguas civilizaciones griega y romana. Este dispositivo es muy sencillo, consta de cuentas ensartadas en varillas que a su vez están montadas en un marco rectangular. Al desplazar las cuentas sobre varillas, sus posiciones representan valores almacenados, y es mediante dichas posiciones que este representa y almacena datos. A este dispositivo no se le puede llamar computadora por carecer del elemento fundamental llamado programa.

Otro de los inventos mecánicos fue la Pascalina inventada por Blaise Pascal (1623 - 1662) de Francia y la de Gottfried Wilhelm von Leibniz (1646 - 1716) de Alemania. Con estas máquinas, los datos se representaban mediante las posiciones de los engranajes, y los datos se introducían manualmente estableciendo dichas posiciones finales de las ruedas, de manera similar a como leemos los números en el cuentakilómetros de un automóvil.

La primera computadora fue la máquina analítica creada por Charles Babbage, profesor matemático de la Universidad de Cambridge en el siglo XIX. La idea que tuvo Charles Babbage sobre un computador nació debido a que la elaboración de las tablas matemáticas era un proceso tedioso y propenso a errores. En 1823 el gobierno británico lo apoyo para crear el proyecto de una máquina de diferencias, un dispositivo mecánico para efectuar sumas repetidas.

Mientras tanto Charles Jacquard (francés), fabricante de tejidos, había creado un telar que podía reproducir automáticamente patrones de tejidos leyendo la información codificada en patrones de agujeros perforados en tarjetas de papel rígido. Al enterarse de este método Babbage abandonó la máquina de diferencias y se dedicó al proyecto de la máquina analítica que se pudiera programar con tarjetas perforadas para efectuar cualquier cálculo con una precisión de 20 dígitos. La tecnología de la época no bastaba para hacer realidad sus ideas.

El mundo no estaba listo, y no lo estaría por cien años más.

En 1944 se construyó en la Universidad de Harvard, la Mark I, diseñada por un equipo encabezado por Howard H. Aiken. Esta máquina no está considerada como computadora electrónica debido a que no era de propósito general y su funcionamiento estaba basado en dispositivos electromecánicos llamados relevadores.

En 1947 se construyó en la Universidad de Pennsylvania la ENIAC (Electronic Numerical Integrator And Calculator) que fue la primera computadora electrónica, el equipo de diseño lo encabezaron los ingenieros John Mauchly y John Eckert. Esta máquina ocupaba todo un sótano de la Universidad, tenía más de 18 000 tubos de vacío, consumía 200 KW de energía eléctrica y requería todo un sistema de aire acondicionado, pero tenía la capacidad de realizar cinco mil operaciones aritméticas en un segundo.

El proyecto, auspiciado por el departamento de Defensa de los Estados Unidos, culminó dos años después, cuando se integró a ese equipo el ingeniero y matemático húngaro John von Neumann (1903 - 1957). Las ideas de von Neumann resultaron tan fundamentales para su desarrollo posterior, que es considerado el padre de las

computadoras.

La EDVAC (Electronic Discrete Variable Automatic Computer) fue diseñada por este nuevo equipo. Tenía aproximadamente cuatro mil bulbos y usaba un tipo de memoria basado en tubos llenos de mercurio por donde circulaban señales eléctricas sujetas a retardos.

La idea fundamental de von Neumann fue: permitir que en la memoria coexistan datos con instrucciones, para que entonces la computadora pueda ser programada en un lenguaje, y no por medio de alambres que eléctricamente interconectaban varias secciones de control, como en la ENIAC.

Todo este desarrollo de las computadoras suele divisarse por generaciones y el criterio que se determinó para determinar el cambio de generación no está muy bien definido, pero resulta aparente que deben cumplirse al menos los siguientes requisitos: La forma en que están construidas. Forma en que el ser humano se comunica con ellas.

Historia de la Programación

La historia de la Programación está relacionada directamente con la aparición de los computadores, que ya desde el siglo XV tuvo sus inicios con la construcción de una máquina que realizaba operaciones básicas y raíces cuadradas (Gottfried Wilheml von Leibniz); aunque en realidad la primera gran influencia hacia la creación de los computadores fue la máquina diferencial para el cálculo de polinomios, proyecto no concluido de Charles Babbage (1793-1871) con el apoyo de Lady Ada Countess of Lovelace (1815-1852), primera persona que incursionó en la programación y de quien proviene el nombre del lenguaje de programación ADA creado por el DoD (Departamento de defensa de Estados Unidos) en la década de 1970.

Luego los avances en las ciencias informáticas han sido muy acelerados, se reemplazó los tubos al vacío por transistores en 1958 y en el mismo año, se sustituyeron por circuitos integrados, y en 1961 se miniaturizaron en chips de silicio. En 1971 apareció el primer microprocesador de Intel; y en 1973 el primer sistema operativo CP/M. El primer computador personal es comercializado por IBM en el año 1980.

De acuerdo a este breve viaje por la historia, la programación está vinculada a la aparición de los computadores, y los lenguajes tuvieron también su evolución. Inicialmente se programaba en código binario, es decir en cadenas de 0s y 1s, que es el lenguaje que entiende directamente el computador, tarea extremadamente difícil; luego se creó el lenguaje ensamblador, que aunque era lo mismo que programar en binario, al estar en letras era más fácil de recordar. Posteriormente aparecieron lenguajes de alto nivel, que en general, utilizan palabras en inglés, para dar las órdenes a seguir, para lo cual utilizan un proceso intermedio entre el lenguaje máquina y el nuevo código llamado código fuente, este proceso puede ser un compilador o un intérprete.

Un compilador lee todas las instrucciones y genera un resultado; un intérprete ejecuta y genera resultados línea a línea. En cualquier caso han aparecido nuevos lenguajes de programación, unos denominados estructurados y en la actualidad en cambio los lenguajes orientados a objetos y los lenguajes orientados a eventos.

Tipos de Lenguajes de programación

Los lenguajes de programación se dividen principalmente en dos tipos: los de bajo nivel, que se comunican directamente con el lenguaje binario de las máquinas; y los de alto nivel, que facilitan su comprensión por parte de los programadores.

Actualmente existen muchos tipos de lenguaje de programación que se utilizan dependiendo de los objetivos del software que se quiere desarrollar. Por ejemplo, actualmente la programación web está teniendo gran auge. Por ello, los lenguajes de programación que llamamos del lado del cliente (PHP y Python, por ejemplo) y del lado del servidor (Javascript) son de los más utilizados

1. Java
2.Lenguaje de programación C
3. Python
4. C++
5. C#
6. Visual Basic. NET
7. SQL
8. PHP
9. Ruby
10. Lenguaje de programación R
11. Rust
12. TypeScript
13. Swift
14. Perl
15.Lenguaje de programación Go
16. Kotlin
17. Scheme
18. Erlang
19. Elixir
20. Pascal
21. Postscript
22. Haskell
23. Objective-C
24. Scala
25. Lava

Clasificación de los lenguajes de programación

Lenguajes Imperativos

Su origen es la propia arquitectura de von Neumann, que consta de una secuencia de celdas (memoria) en las cuales se pueden guardar datos e instrucciones, y de un procesador capaz de ejecutar de manera secuencial una serie de operaciones (ó comandos) principalmente aritméticas y booleanas. En general, un lenguaje imperativo ofrece al programador conceptos que se traducen de forma natural al modelo de la máquina. Ejemplos: FORTRAN, Algol, Pascal, C, Modula-2, Ada.

El programador tiene que traducir la solución abstracta del problema a términos muy primitivos, cercanos a la máquina, por lo que los programas son más "comprensibles" para la máquina que para el hombre. Esto es una desventaja para nosotros que hace que sea sumamente complicado construir código en lenguaje imperativo. Lo bueno de este lenguaje es que es tan cercano al lenguaje de la máquina que la eficiencia en la ejecución es altísima.

Lenguajes Funcionales

Los matemáticos resuelven problemas usando el concepto de función, que convierte datos en resultados. Sabiendo cómo evaluar una función, usando la computadora, podríamos resolver automáticamente muchos problemas. Este fue el pensamiento que llevó a la creación de los lenguajes de programación funcionales. Además, se aprovechó la posibilidad que tienen las funciones para manipular datos simbólicos, y no solamente numéricos, y la propiedad de las funciones que les permite componer, creando de esta manera, la oportunidad para resolver problemas complejos a partir de las soluciones a otros más sencillos. También se incluyó la posibilidad de definir funciones recursivamente. Un lenguaje funcional ofrece conceptos que son muy entendibles y relativamente fáciles de manejar.

El lenguaje funcional más antiguo y popular es LISP, diseñado por McCarthy en la segunda mitad de los años 50. Se usa principalmente en Inteligencia Artificial. En los 80 se añadió a los lenguajes funcionales la tipificación y algunos conceptos modernos de modularización y polimorfismo, un ejemplo es el lenguaje ML.

Programar en un lenguaje funcional significa construir funciones a partir de las ya existentes. Por lo tanto es importante conocer y comprender bien las funciones que conforman la base del lenguaje, así como las que ya fueron definidas previamente. De esta manera se pueden ir construyendo aplicaciones cada vez más complejas. La desventaja es que está alejado del modelo de la máquina de von Neumann y, por lo tanto, la eficiencia de ejecución de los intérpretes de lenguajes funcionales es peor que la ejecución de los programas imperativos precompilados.

Lenguajes Lógicos

Otra forma de razonar para resolver problemas en matemáticas se fundamenta en la lógica de primer orden. El conocimiento básico de las matemáticas se puede representar en la lógica en forma de axiomas, a los cuales se añaden reglas formales para deducir cosas verdaderas (teoremas). Gracias al trabajo de algunos matemáticos, de finales de siglo pasado y principios de éste, se encontró la manera de automatizar computacionalmente el razonamiento lógico -particularmente para un subconjunto significativo de la lógica de primer orden- que permitió que la lógica matemática diera origen a otro tipo de lenguajes de programación, conocidos como lenguajes lógicos. También se conoce a estos lenguajes, y a los funcionales, como lenguajes declarativos, porque para solucionar un problema el programador solo tiene que describirlo con axiomas y reglas de deducción en el caso de la programación lógica y con funciones en el caso de la programación funcional.

En los lenguajes lógicos se utiliza el formalismo de la lógica para representar el conocimiento sobre un problema y para hacer preguntas que se vuelven teoremas si se demuestra que se pueden deducir a partir del conocimiento dado en forma de axiomas y de las reglas de deducción estipuladas. Así se encuentran soluciones a problemas formulados como preguntas. Con base en la información expresada dentro de la lógica de primer orden, se formulan las preguntas sobre el dominio del problema y el intérprete del lenguaje lógico trata de encontrar la respuesta automáticamente. El conocimiento sobre el problema se expresa en forma de predicados (axiomas) que establecen relaciones sobre los símbolos que representan los datos del dominio del problema. El PROLOG surgió a principio de los 70 y es el primer lenguaje lógico. Las aplicaciones en la Inteligencia Artificial lo mantienen vivo y útil.

En el caso de la programación lógica, el trabajo del programador es la buena descripción del problema en forma de hechos y reglas. A partir de ésta se pueden encontrar muchas soluciones dependiendo de como se formulen las preguntas (metas), que tienen sentido para el problema. Si el programa está bien definido, el sistema encuentra automáticamente las respuestas a las preguntas formuladas. En este caso ya no es necesario definir el algoritmo de solución, como en la programación imperativa, lo fundamental aquí es expresar bien el conocimiento sobre el problema.

En programación lógica, al igual que en programación funcional, el programa, en este caso los hechos y las reglas, están muy alejados del modelo von Neumann que posee la máquina en la que tienen que ser interpretados; por lo que la eficiencia de la ejecución es inferior a la de un programa equivalente en lenguaje imperativo. Sin embargo, para cierto tipo de problemas, la formulación del programa mismo puede ser mucho más sencilla y natural.

Lenguajes Orientados a Objetos

A mediados de los años 60 se empezó a usar las computadoras para la simulación de problemas del mundo real. Pero el mundo real está lleno de objetos, en la mayoría de los casos complejos, los cuales difícilmente se traducen a los tipos de datos primitivos de los lenguajes imperativos. Así surgió el concepto de objeto y sus colecciones (clases de objetos), que permitieron introducir abstracciones de datos a los lenguajes de programación. La posibilidad de reutilización del código y sus indispensables

modificaciones, se reflejaron en la idea de las jerarquías de herencia de clases. También surgió el concepto de polimorfismo introducido vía procedimientos virtuales. Todos estos conceptos (que hoy identificamos como conceptos del modelo de objetos) fueron presentados en el lenguaje Simula 67, desde el año 1967, aunque este lenguaje estaba enfocado a aplicaciones de simulación discreta. Fue en los años 80 cuando surgieron lenguajes de programación con conceptos de objetos encabezada por Smalltalk, C++, Eiffel, Modula-3, Ada 95 y terminando con Java.

El modelo de objetos, y los lenguajes que lo usan, parecen facilitar la construcción de sistemas o programas en forma modular. Los objetos ayudan a expresar programas en términos de abstracciones del mundo real, lo que aumenta su comprensión. La clase ofrece cierto tipo de modularización que facilita las modificaciones al sistema. La reutilización de clases previamente probadas en distintos sistemas también es otro punto a favor. Sin embargo, el modelo de objetos, a la hora de ser interpretado en la arquitectura von Neumann conlleva un excesivo manejo dinámico de memoria debido a la constante creación de objetos, así como a una carga de código fuerte causada por la constante invocación de métodos. Por lo tanto los programas en lenguajes orientados a objetos son ineficientes, en tiempo y memoria, contra los programas

equivalentes en lenguajes imperativos, aunque les ganan en la comprensión de código.

Lenguajes Concurrentes, Paralelos y Distribuidos

El origen de los conceptos para el manejo de concurrencia, paralelismo y distribución está en el deseo de aprovechar al máximo la arquitectura von Neumann y sus modalidades reflejadas en conexiones paralelas y distribuidas.

Esto fue un tema importante sobre todo cuando las computadoras eran caras y escasas; el sistema operativo tenía que ofrecer la ejecución concurrente y segura de programas de varios usuarios, que desde distintos terminales utilizaban un solo procesador, y así surgió la necesidad de introducir algunos conceptos de programación concurrente para programar los sistemas operativos.

Cuando los procesadores cambiaron de tamaño y de precio, se abrió la posibilidad de contar con varios procesadores en una máquina y ofrecer el procesamiento en paralelo (procesar varios programas al mismo tiempo). Esto dio el impulso a la creación de lenguajes que permitían expresar el paralelismo. Finalmente, llegaron las redes de computadoras, que también ofrecen la posibilidad de ejecución en paralelo, pero con procesadores distantes, lo cual conocemos como la programación distribuida.

Históricamente encontramos soluciones conceptuales y mecanismos (semáforos, regiones críticas, monitores, envío de mensajes, llamadas a procedimientos remotos (RPC)) que se incluyeron después en lenguajes de programación como Concurrent Pascal o Modula (Basados en monitores), Ada ó SR (Basada en RendezVous (Tipo deRPC)), ALGOL 68(Semáforos), OCCAM (Envío de mensajes), Java...

Otros tipos de lenguajes de programación son:

Procedural Language, Declarative Language, Applicative Language, Definitional Language, Single Assignment Language, Dataflow Language, Constraint Language, Lenguaje de cuarta generación(4GL), Query Language, Specification Language, Assembly Language, Intermediate Language, Metalenguajes.

Tipos de paradigmas de programación

Un paradigma de programación es una manera o estilo de programación de software. Existen diferentes formas de diseñar un lenguaje de programación y varios modos de trabajar para obtener los resultados que necesitan los programadores. Se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas computacionales.

Paradigma imperativo

Los programas consisten en una sucesión de instrucciones o conjunto de sentencias, como si el programador diera órdenes concretas. El desarrollador describe en el código paso por paso todo lo que hará su programa.

Algunos lenguajes: Pascal, COBOL, FORTRAN, C, C++, etc.

Otros enfoques subordinados al paradigma de programación imperativa son:

- Programación estructurada: La programación estructurada es un tipo de programación imperativa donde el flujo de control se define mediante bucles anidados, condicionales y subrutinas, en lugar de a través de GOTO.

- Programación procedimental: Este paradigma de programación consiste en basarse en un número muy bajo de expresiones repetidas, englobarlas todas en un procedimiento o función y llamarlo cada vez que tenga que ejecutarse.

- Programación modular: consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más manejable y legible. Se trata de una evolución de la programación estructurada para resolver problemas de programación más complejos.

Paradigma declarativo

Este paradigma no necesita definir algoritmos puesto que describe el problema en lugar de encontrar una solución al mismo. Este paradigma utiliza el principio del razonamiento lógico para responder a las preguntas o cuestiones consultadas.

Este paradigma a su vez se divide en dos:

- Programación Lógica: Prolog
- Programación funcional: Lisp, Scala, Java, Kotlin.

Programación orientada a objetos

En este modelo de paradigma se construyen modelos de objetos que representan elementos (objetos) del problema a resolver, que tienen características y funciones. Permite separar los diferentes componentes de un programa, simplificando así su creación, depuración y posteriores mejoras. La programación orientada a objetos disminuye los errores y promociona la reutilización del código. Es una manera especial de programar, que se acerca de alguna manera a cómo expresaríamos las cosas en la vida real.

Podemos definir un objeto como una estructura abstracta que, de manera más fiable, describe un posible objeto del mundo real y su relación con el resto del mundo que lo rodea a través de interfaces. Ejemplos de lenguajes de programación orientados a objetos serían Java, Python o C#.

La programación orientada a objetos se sirve de diferentes conceptos como:

- Abstracción de datos
- Encapsulación
- Eventos
- Modularidad
- Herencia
- Polimorfismo

Programación reactiva

Este Paradigma se basa en escuchar lo que emite un evento o cambios en el flujo de datos, en donde los objetos reaccionan a los valores que reciben de dicho cambio. Las librerías más conocidas son Project Reactor, y RxJava. React/Angular usan RxJs para hacer uso de la programación reactiva.