Historia de los Lenguajes de Programación

Los primeros lenguajes de programación surgen a mediados del siglo XIX, de la idea de un profesor matemático de la Universidad de Cambridge llamado Charles Babagge. Consistían en lo que él denominaba máquina analítica, la que recién pudo construirse a mediados del siglo XX. Con él colaboró Ada Lovedby, quien es considerada la primera programadora de la historia, por desarrollar programas para la máquina analítica de Babagge en tarjetas perforadas.

Babagge intento crear una máquina que pudiera programar con tarjeras perforadas que efectuara cualquier cálculo con una precisión de 20 dígitos, pero la tecnología de la época no bastaba para hacer realizar sus ideas, por lo que no llegaron a materializarse. Sin embargo su contribución fue decisiva ya que las computadoras actuales responden al esquema de la máquina analítica. El diseño propuesto por Babagge cuenta con cinco unidades:

- Unidad de entrada, para introducir datos e instrucciones.
- Memoria, donde se almacenaban datos y resultados intermedios.
- Unidad de control, para regular la secuencia de ejecución de las operaciones.
- Unidad Aritmético−Lógica, que efectúa las operaciones.
- Unidad de salida, encargada de comunicar al exterior los resultados.

Siguiendo la idea de Babagge se construyeron las primeras computadoras, entre las cuales se destaca ENIAC (Electronic Numerical Integrator and Calculator), su programación se basaba en componentes físicos, es decir que se programaba cambiando directamente el Hardware de la máquina, esto significa, cambiar cables de lugar para así conseguir la programación de la máquina. La entrada y salida de datos se realizaba utilizando tarjetas perforadas.

En la década de los 40 se crearon las primeras computadoras modernas, con alimentación eléctrica, la velocidad y capacidad de memoria limitada forzaron a los programadores a escribir programas en lenguaje ensamblador.

Primeros Lenguajes

Los primeros lenguajes de programación denominados código o lenguaje de máquina se basaban en código binario y estaban especializados según sus aplicaciones. Pero este tipo de programación es muy lenta y tediosa, debido a que tanto datos como las instrucciones deben introducirse al sistema binario, además se debe conocer y acceder a posiciones de memoria. En este tipo de programación surge gran cantidad de errores, lo que significa que la tarea de depuración demandará mucho tiempo y dedicación.

A principios de los 50 aparece una nueva notación simbólica denominada código de ensamblaje, utilizaba una serie de abreviaturas para representar las diferentes operaciones: ADD (suma), STORE (copiar), etc. La traducción del código máquina al código de ensamblaje se realizaba manualmente, luego se observó que la computadora podía realizar esta tarea, entonces de desarrollo un programa traductor, denominado Ensamblador [] (ASSEMBLER).

Conforme las computadoras se introducían en las empresas y establecimientos educativos, los lenguajes primitivos fueron sustituidos por otros más sencillos de emplear y fáciles de aprender. Estos lenguajes denominados de alto nivel, poseen una estructura que se adapta mejor al pensamiento humano.

El primer lenguaje de alto nivel fue FORTRAN (FORmula TRANslation) desarrollado entre 1954 y 1957 por un equipo de IBM, diseñado para programación científica y de computación. Este lenguaje fue utilizado en programación de tipo general, con el transcurso del tiempo se incorporaron nuevas características tomadas de otros lenguajes (FORTRAN II, FORTRAN IV, FORTRAN 66, entre otros.) Su supervivencia talvez se deba a que los compiladores para este lenguaje son de los más eficientes que existen generando código muy rápido. Por ser el primer lenguaje de alto nivel, posee características novedosas las que luego se convirtieron en estándares para los lenguajes posteriores.

COBOL y Algol60 fueron lenguajes que tuvieron un gran impacto en la programación y en el uso de computadoras. COBOL (COmmon Business-Oriented Language) fue desarrollado por el Departamento de Defensa de Estados Unidos (1959-1960), fue adoptado rápidamente por bancos y empresas para el desarrollo de operaciones a gran escala y otras aplicaciones comerciales. Algunas de sus características por las cuales COBOL se considera pionero son: estructura de registros para la organización de datos, estructura de datos separada de la sección de ejecución, formato versátil de salida utilizando imágenes o ejemplos de salida.

Algol60 (ALGOrithmic Language) fue desarrollado para proporcionar un lenguaje general y expresivo para la descripción de algoritmos, tanto en la investigación como en las aplicaciones prácticas. La mayoría de los lenguajes imperativos actuales derivan de Algol, como por ejemplo Pascal, C y Ada. Algol incorporó nuevos conceptos en la programación, incluyendo enunciados estructurados de formato libre, bloques de inicio y fin, declaración de tipos para las variables, recursión y paso de parámetros de pasar por valor. También introdujo implícitamente el ambiente de tiempo de ejecución basado en pilas para los lenguajes estructurados, que se utiliza hasta la actualidad y fue el primero en utilizar la notación con formato Backus-Naur (BNF) para definir la sintaxis.

Al mismo tiempo que se creaban los lenguajes mencionados, se desarrollaban otros basados en conceptos matemáticos de la función. Dos ejemplos de estos lenguajes son LISP y APL.

Los años 60

En esta década se vio el desarrollo de cientos de lenguajes de programación, cada uno de ellos incorporaba los intereses o preocupaciones particulares de sus diseñadores (lenguajes de propósito especial). La mayoría de estos lenguajes desapareció, unos pocos tuvieron un efecto significativo en la programación, en la siguiente tabla se listan algunos de los lenguajes que existían en 1967.

ADAM	COMIT	GECOM	NELIAC
AED	CORAL	GPL	OCAL
AESOP	CORC	GPSS	OMNITAB
AIMACO	CPS	GRAF	OPS
ALGOL	DAS	GRAF	OPS
ALGY	DATA-TEXT	IDS	PENCIL
ALTRAN	DIALOG	IT	PRINT
AMBIT	DEACON	JOSS	QUIKTRAN
AMTRAN	DIAMAG	JOVIAL	SFD-ALGOL
APL	DIMATE	L	SIMSRIPT
BACIAC	DOCUS	LDT	SIMULA
BASEBALL	DSL	LISP	SNOBOL
BASIC	DYANA	LOLITA	SOL
BUGSYS	DYNAMO	LOTIS	SPRINT
C-10	DYSAC	MAD	STRESS
CLIP	FLAP	MADCAP	STROBES
CLP	FLOW-MATIC	MAP	TMG
COBOL	FORMAC	MATHLAB	TRAC
COGENT	FORTRAN	MATH-MATIC	TRANDIR
COGO	FORTRANSIT	META	TREET
COLASL	FSL	MILITRAN	UNCOL
COLINGO	GAT	MIRFAC	UNICODE

Algunos de los diseñadores de los años 50 comenzaron a pensar en lenguajes más generales y universales, tal vez en uno que terminará con todos los demás. Este proyecto coincidía con el PL/I de IBM (1963-1964) destinado a ser utilizado por una única familia de computadoras, combinando las características de FORTRAN, COBOL y Algol60 incorporando el manejo de concurrencia y excepciones. Este proyecto fracasó, los traductores resultaron difíciles de escribir, lentos, grandes y nada seguros, además de difícil de aprender y propenso a errores. Se piensa que PL/I fue creado antes de tiempo, puesto que conceptos tales como concurrencia y excepciones no eran comprendidos claramente en esa época. Algo similar ocurrió con Algol68, una versión mejorada de Algol60 con una estructura más expresiva y teóricamente más consistente.

No todos los lenguajes desarrollados en los 60 fueron un fracaso, muchos fueron ampliamente utilizados y ofrecieron contribuciones significativas y duraderas al desarrollo

de lenguajes de programación. Un ejemplo de estos es SNOBOL (StriNg Oriented symBOlic Language) uno de los primeros lenguajes de procesamiento de cadenas, en su versión SNOBOL4 proporciona herramientas de coincidencia de patrones complejas y poderosas.

Otro lenguaje que se destacó es Simula67, originalmente pensado para simulaciones contribuyó de manera fundamental a la comprensión de la abstracción y de la computación a través de su introducción del concepto de Clase de los lenguajes orientados a objetos.

El lenguaje ISWIM también desarrollado en los 60, influyó aún más que LISP en los lenguajes funcionales modernos como ML y Haskell, de hecho fue el primer lenguaje basado completamente en formalismos matemáticos y cuyo comportamiento se describió con total precisión. Por último otro ejemplo de esta década es BASIC (1964) su finalidad era ser un lenguaje simple para los nuevos sistemas de tiempo compartido de la época. Diez años después sufrió una transición natural hacia las microcomputadoras y sigue utilizándose en la actualidad. BASIC se considera una familia de lenguajes.

Los años 70 : simplicidad, abstracción, estudio.

Luego de la confusión de los años 60, los diseñadores volvieron hacia atrás en sus proyectos, apreciando la simplicidad y consistencia del diseño de lenguajes. Algol 68 recibió un riguroso rechazo por parte de Wirth y otros, quienes publicaron Algol-W y en 1971 escribió el lenguaje de programación Pascal, refinando las ideas de Algol en un lenguaje pequeño, simple, eficiente y estructurado. Pascal obtuvo éxito y aceptación al punto de ser utilizado no solo en la enseñanza de programación, sino también usos prácticos, a pesar de omitir características importantes como compilación independiente, manejo adecuado de cadenas y capacidades de entrada y salida.

En 1972, Ritchie desarrolló el lenguaje de programación C, que intenta alcanzar la simplicidad de Pascal al conservar y restringir la orientación de la expresión, reducir la complejidad del sistema de tipos y el entorno en tiempo de ejecución y proporcionar mas acceso a la máquina subyacente.

A mediados y fines de los 70, los diseñadores experimentaron con mecanismos de abstracción de datos, concurrencia y verificación (programa correcto).

Los años 80

Fueron años de consolidación relativa en los lenguajes imperativos, se comenzó a trabajar sobre las ideas de la década anterior en lugar de inventar nuevos lenguajes. Se invirtieron grandes sumas de dinero en la investigación de los denominados lenguajes de quinta generación con la incorporación de construcciones de la programación lógica. En esta década tuvo un fuerte crecimiento la programación orientada a objetos, lo que dio origen a

la creación de muchos lenguajes tales como Objective C, Object Pascal, Modula-3, Oberon y Eiffel.

El diseño de lenguajes se enfocaba en la programación de sistemas a gran escala de unidades de código, los sistemas de módulos eran relacionados con frecuencia con construcciones de programación genérica, en esencia eran módulos parametrizados.

Los años 90

El gran acontecimiento histórico de esta década fue el crecimiento de Internet, lo que provocó que el mercado de las computadoras personales creciera notablemente, al igual que la expansión de los sistemas operativos basados en ventanas. Hasta 1993 C++ "ganaba" la guerra de los lenguajes de programación orientados a objetos, hasta 1995 cuando aparece Java.

Java fue desarrollado por James Gosling en Sun Microsystems, no para Internet sino para aplicaciones electrónicas incrustadas. Ante la popularidad de Internet los desarrolladores de Sun lo revisaron y adaptaron para utilizarlos en la Web y en la red. Algunas de las características que favorecieron su aceptación que se trata de un lenguaje relativamente simple, limpiamente diseñado y provisto de una biblioteca de herramientas para ventanas, redes y concurrencia. Además de C++ y Java, otros lenguajes alcanzaron un grado de madurez significativo, ellos son Haskell y el nuevo estándar Ada95. Un problema que los lenguajes desarrollados en los 90 han puesto de manifiesto es el de las bibliotecas, hasta entonces consideradas secundarias, no formaban parte del lenguaje de programación. Con el aumento de la importancia de las bibliotecas, surgen los lenguajes de scripts, incluyendo a AWK, Perl, Tcl, Javascripts, Rexx y Python. Las características de estos lenguajes son: programas breves, comportamiento altamente dinámico, estructuras de alto nivel de datos incorporados, una sintaxis extensible y carencia de verificación de tipos.

Tendencias actuales

Los lenguajes de programación continúan en evolución, tanto en la investigación como en la industria, algunos de las tendencias actuales incluyen:

Desarrollo de software orientado a componentes

Construir lenguajes que apoyen la programación concurrente y distribuida.

Programación orientada a aspectos (POA)

Integración con bases de datos, incluyendo XML y bases de datos relacionales

Mecanismos que aporten verificación al lenguaje, en cuanto a seguridad y confiabilidad

Entre los lenguajes desarrollados a partir del año 2000 se destacan: ActionScript , C# , D y Visual Basic Net, Groovy, Grace, Assembly language, Python, Ruby, etc.

Aspectos que provocan la evolución de los Lenguajes de Programación

- Recursos y tipos de computadoras
- Aplicaciones y necesidades de los usuarios
- Nuevos métodos de programación
- Estudios teóricos
- Estandarización

El estudio de los Lenguajes de Programación mejora el uso de los mismos, incrementa el vocabulario de los elementos de programación, permite una mejor elección de los LP, mejora la habilidad de desarrollo de programas efectivos y eficientes, además de facilitar el aprendizaje y el diseño de nuevos LP.

Definiciones

Un lenguaje de programación es un conjunto de caracteres, reglas para su combinación y reglas especificando sus efectos cuando son ejecutadas por una computadora, las cuales deben cumplir con las siguientes características:

- *No requiere del conocimiento del código máquina por parte del usuario*
- *Es independiente de la máquina*
- *Es posible traducirlo a lenguaje máquina*
- *Es cercano al problema específico que se resuelve en el lenguaje de máquina*[1]

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 de precisión o como medio de comunicación humana.
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 (cuando es necesario) y de mantiene el código fuente de programa informático se llama programación.

Características y elementos de los Lenguajes de Programación

- Definen un proceso que se ejecuta en una computadora.
- Es de alto nivel, cercano a los problemas que se requieren resolver (abstracción)
- Permite construir nuevas abstracciones que se adapten al dominio que se programa.

Los LP permite combinar ideas simples en ideas más complejas de acuerdo a los siguientes tres mecanismos: expresiones primitivas (unidades más simples), mecanismos de

[1] Definición de la Encyclopedia of Computer Science

combinación, que permiten construir elementos compuestos a partir de unidades simples y mecanismos de abstracción, con los cuales permite dar nombre a los elementos compuestos y manipularlos como unidades.

Paradigmas de Programación

Un Paradigma se define como un conjunto de reglas, patrones y estilos de programación usados por un grupo de LP. Existen diferentes, cada uno de ellos tiene sus propias características y tratan de solucionar los problemas clásicos del desarrollo de software.

Paradigma imperativo: en este paradigma se describen las sentencias que modifican el estado de un programa. Se expresa cómo debe solucionarse un problema especificando una secuencia de acciones a realizar a través de uno o más procedimientos denominadas subrutinas o funciones.

Programación estructurada: los datos y procedimientos están separados y sin relación, lo que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. El programador piensa todo en término de funciones y procedimientos, en segundo lugar en la estructura de los datos.

Paradigma declarativo: se contrapone a la programación imperativa, en este paradigma no es necesario definir un algoritmo, debido a que se detalla la solución del problema el lugar de **como** llegar a esa solución. La solución de problemas específicos se alcanza a través de mecanismos internos de control, pero no se especifica exactamente. Las variables se utilizan con transparencia referencial, es decir, una expresión puede ser sustituida por el resultado de ser evaluada en el programa, sin alterar su semántica. Los siguientes son sub-paradigmas derivados del paradigma declarativo:

Paradigma funcional: la computadora evalúa expresiones y funciones matemáticas.

```
>>> def incrementar(x):
...     return x + 1
>>> numeros = map(incrementar,[2,4,10,30,50])
>>> print numeros
[3, 5, 11, 31, 51]
>>> max([5,3,2,6,8])
8
>>> min([5,3,2,6,8])
2
```

Paradigma lógico: se definen reglas, se basa en el uso de sentencias lógicas para representar y evaluar programas.

```
for letra in nombre:
    vocales += 1 if letra in 'aeiou' else 0
a = True ; b = False ; c = True ; d = True
if a and b and c and d: print 'Todo cierto'
if all([a,b,c,d]): print 'Todo cierto'
if a or b or c or d: print 'Al menos uno es cierto'
if any([a,b,c,d]): print 'Al menos uno es cierto'
```

Paradigma orientado a objetos: los programas se definen en términos de clases de objetos, objetos que combinan estados, comportamiento e identidad. Este paradigma expresa los programas como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Permite que los programas y módulos sean fáciles de escribir, mantener y reutilizar. La programación orientada a objetos, posee las siguientes características principales:

Encapsulamiento: ubica en el mismo nivel de abstracción a todos los elementos, estados y comportamiento, que pueden considerarse pertenecientes a una misma entidad. Permite aumentar la cohesión.

Herencia: las clases se relacionan entre sí dentro de una jerarquía de clasificación que permite definir nuevas clases basadas en clases preexistentes, permitiendo crear objetos especializados.

Polimorfismo: comportamiento diferente basado en la misma denominación, asociados a objetos distintos entre sí.

Paradigmas actuales

Este momento de la programación se caracteriza por la combinación de varios paradigmas en un mismo lenguaje:

Programación lógico-funcional: Curry, TOY

Programación funcional-concurrente: Erlang (80’), Concurrent Haskell (87’- 10’)

Programación funcional-paralela: GpH, Eden (90’), Data Parallel Haskell (02’)

Programación lógico-funcional-concurrente-orientado a objetos: Oz (99 - 05’)

A su vez los lenguajes orientados a objetos incorporan construcciones funcionales:

JavaScript: maps, folds, closures

Java 8: maps, folds, lambda, abstractions

Ruby: lambda abstractions, closures, firts-class, continuations

Python: map, reduce, filters, list comprehensions, lambda abstractions

Los lenguajes funcionales incorporan facilidades para la orientación a objetos: Scala, F#. La mayoría de los lenguajes de script son interpretados y tienen tipado dinámico, entre los cuales se mencionan: Ruby, PHP, Python y JavaScripts entre otros.

Inconvenientes:

- Requieren más memoria y más tiempo para comprobar tipos en ejecución
- Se detectan errores de tipos en los caminos realmente ejecutados

Estos lenguajes convierten automáticamente unos tipos con otros con excesiva flexibilidad. (Python mediante la asignación)

Esta flexibilidad, sumada a la mezcla de paradigmas y amplias facilidades de metaprogramación que soportan, advierten que estos lenguajes no deberían ser utilizados por programadores no expertos.

Los lenguajes de programación han evolucionado enfocándose en los siguientes aspectos:

- Creciente nivel de abstracción: tipos definidos por el usuario, clases, funciones de orden superior, construcciones para la concurrencia entre otros.
- Mecanismos de modularidad: permiten independizar una parte del programa de otras.
- Mecanismos de seguridad estática: los sistemas de tipos protegen de errores triviales que de otro modo pasarían inadvertidos. Suficiente flexibilidad para el programador.
- Reutilización: los tipos como el polimorfismo, la sobrecarga, la genericidad, la modularidad y el orden superior potencian el desarrollo de componentes reutilizables.