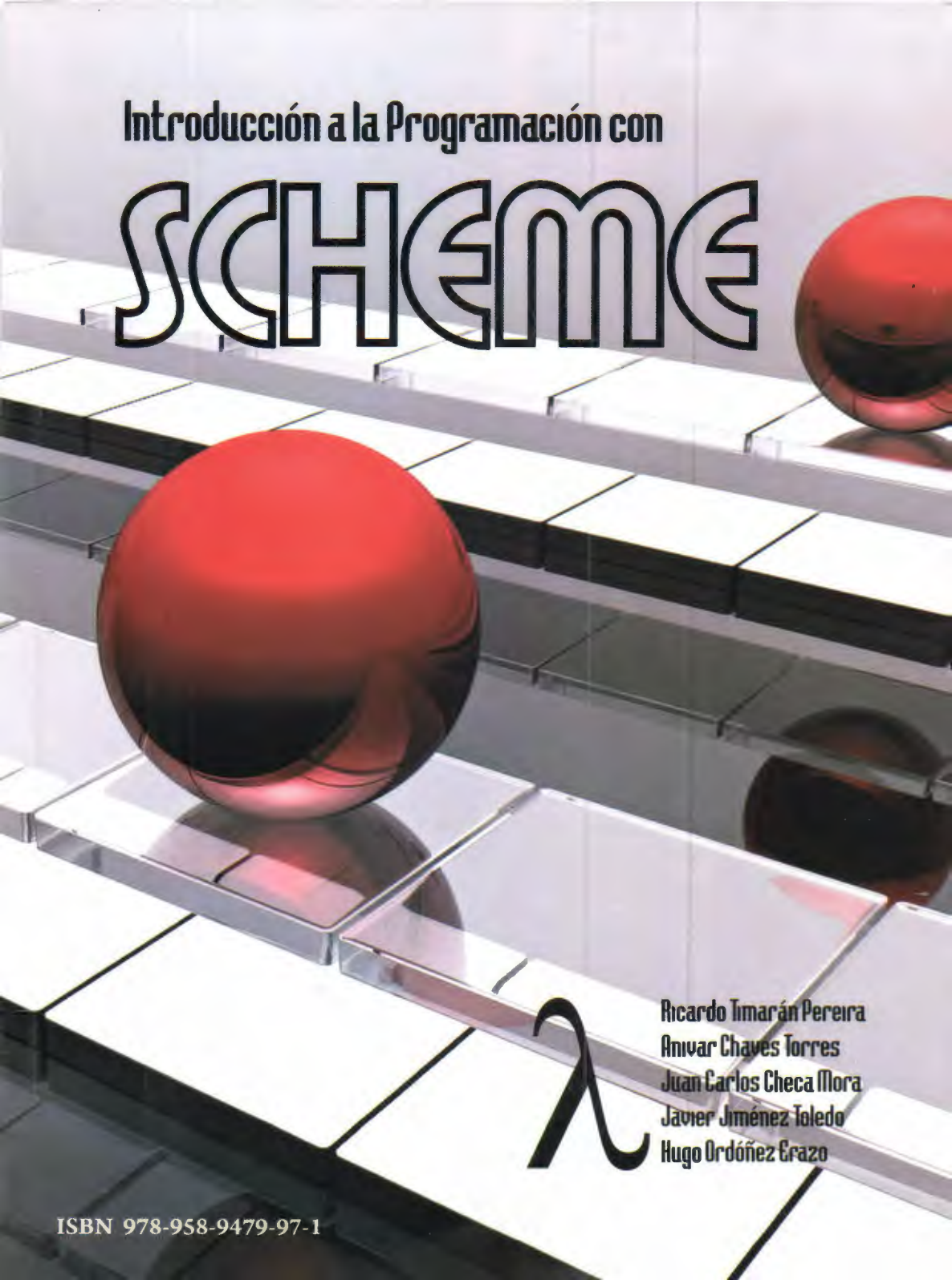
Introducción a la Programación con SCHEME



Ricardo Timarán Pereira
Anivar Chaves Torres
Juan Carlos Checa Mora
Javier Jiménez Toledo
Hugo Ordóñez Erazo

ISBN 978-958-9479-97-1

Introducción a la Programación con Scheme

Ricardo Timarán Pereira
Anívar Chaves Torres
Juan Carlos Checa Mora
Javier Jiménez Toledo
Hugo Ordóñez Erazo

Red Universitaria de Investigación en Sistemas de Nariño
RUISNAR

Red de Universidades Regionales Latinoamericanas
UREL
Capítulo Nariño

ISBN: 978 - 958 - 9479 - 97 - 1

Edición y diagramación: Anívar Chaves Torres

Impreso por: Editorial Universidad de Nariño

A mi querida esposa Amanda
y a mis hijas Andrea, Sofía y Nataly.
Ricardo

A Diana Sofía, Jacky y Camilo
por su confianza y su comprensión.
Anívar

A mi hijo Julián Antonio
de quien aprendo, a quien enseño
y con quien construyo un nuevo mundo.
Juan Carlos

a Dios por ser mi guía,
a Davis, Santiago y Alejandro
quienes le dan sentido a mi vida,
a Ximena por su leal compañía,
a mis padres y hermanos
por su incondicional apoyo.
Javier

A DIOS,
fuente de toda sabiduría y conocimiento
y a mi madre por darme la vida.
Hugo

RED DE UNIVERSIDADES REGIONALES LATINOAMERICANAS
UREL
Capítulo Nariño

Carlos Folleco Erazo
Rector Universidad Cooperativa de Colombia
Presidente Red UREL Capítulo Nariño

Padre Evaristo Acosta Maestre
Rector Institución Universitaria CESMAG

Jaime Tito Colunge Benavides
Rector Corporación Universitaria Autónoma de Nariño

Luis Enrique Ortega Ruales
Director Zonal Universidad Nacional Abierta y a Distancia – UNAD

Orlando Rodríguez García
Director Fundación Universitaria San Martín – Sede Pasto

Deyanira Ortega
Directora Universidad Antonio Nariño – Sede Pasto

Gonzalo Hernández Arteaga
Director Regional Universidad Javeriana

Sara Ángela Arturo González
Directora Regional Servicio Nacional de Aprendizaje – SENA

Silvio Sánchez Fajardo
Rector Universidad de Nariño

Hermana Martha Estela Santa Castrillón
Rectora Universidad Mariana

Melba Rivas Benavides
Directora Universidad Santo Tomás Sede Pasto

Luis Antonio Heredia
Coordinador Fundación Universitaria del Área Andina Sede Pasto

Rocío de la Espriella Guerrero
Directora Escuela Superior de Administración Pública – ESAP

Roberto Narváez Moreno
Director Corporación Universitaria Remington – Sede Pasto

Gerardo Mesías Méndez
Director Ejecutivo Cámara de Comercio de Pasto

RED UNIVERSITARIA DE INVESTIGACIONES
RED UREL
Capítulo Nariño

Isabel Hernández Arteaga
Directora de Investigaciones
Universidad Cooperativa de Colombia
Coordinadora Red Universitaria de Investigaciones

Edmundo Apraez Guerrero
Vicerrector de Investigaciones, Postgrados
y Relaciones Internacionales
Universidad de Nariño

María Eugenia Córdoba
Vicerrectora de Investigaciones
Institución Universitaria CESMAG

Roberto García Castaño
Director del Centro de Investigaciones y Publicaciones
Universidad Mariana

Julio Rafael Burbano Erazo
Director de Investigaciones
Corporación Universitaria Autónoma de Nariño

Elehonora Argoty Pérez
Coordinadora de Investigaciones
Fundación Universitaria San Martín

Omar Martínez Roa
Coordinador Zonal de Investigaciones
Universidad Nacional Abierta y a Distancia – UNAD

RED UNIVERSITARIA DE INVESTIGACIÓN EN SISTEMAS DE NARIÑO
RUISNAR

Ricardo Timarán Pereira
Doctor en Ingeniería
Master of Science en Ingeniería
Especialista en Multimedia
Ingeniero de Sistemas y Computación
Universidad de Nariño
Director RUISNAR

Anívar Chaves Torres
Especialista en Docencia Universitaria
Ingeniero de Sistemas
Investigador
Institución Universitaria CESMAG

Juan Carlos Checa Mora
Especialista en Multimedia
Ingeniero de Sistemas
Investigador
Universidad Cooperativa de Colombia

Constanza Colunge
Ingeniera de Sistemas
Investigadora
Corporación Universitaria Autónoma de Nariño

Javier Jiménez Toledo
Especialista en Docencia Universitaria
Ingeniero de Sistemas
Investigador
Universidad de Nariño

Hugo Ordóñez Erazo
Especialista en Gerencia Informática
Ingeniero de Sistemas
Investigador
Universidad Mariana

CONTENIDO

LISTA DE CUADROS

LISTA DE FIGURAS

PRÓLOGO

"No basta saber,
se debe también aplicar.
No es suficiente querer,
se debe también hacer."
J. W. Goethe

Un curso de introducción a la programación tiene como objetivo brindar a los estudiantes los conceptos, métodos y técnicas, básicos y necesarios, para la solución de problemas utilizando un lenguaje de programación enmarcado dentro de un paradigma específico.

A los estudiantes les resulta muy difícil el paso del enunciado del problema al programa. La programación exige, inicialmente, el entendimiento del problema a través de su análisis; luego, la construcción y prueba del algoritmo y, finalmente, la codificación en un lenguaje de programación. En un primer curso de programación, la cuestión ha resolver es cuál es el paradigma y el lenguaje más adecuados para la enseñanza de los fundamentos de programación.

En la actualidad, existe un debate sobre la conveniencia de aplicar un enfoque estructurado, orientado a objetos o funcional para enseñar los fundamentos de la programación en Ingeniería de Sistemas. Los modelos estructurado y orientado a objetos, por ser imperativos, describen la programación en términos del estado del programa y sentencias que cambian dicho estado. El modelo funcional tiene como objetivo obtener programas donde se exprese la solución a través de funciones, en los que no es necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, evitando el concepto de estado del cómputo.

La Red Universitaria de Investigación en Ingeniería de Sistemas de Nariño - RUISNAR, conformada por la Universidad de Nariño, Universidad Mariana, Universidad Cooperativa, Institución Universitaria CESMAG y la Corporación Universitaria Autónoma de Nariño, instituciones con sede en la ciudad de Pasto, capital del departamento de Nariño (Colombia) y con programas de Ingeniería de Sistemas, realizó el proyecto de investigación *Validación del modelo funcional en la enseñanza de la programación con el lenguaje Scheme en programas Ingeniería de Sistemas*, en el que se estudió, documentó, aplicó y evaluó el modelo funcional utilizando el lenguaje Scheme, en una de las asignaturas del componente de programación de Ingeniería de Sistemas, con fin de desarrollar en el estudiante la capacidad de entender y utilizar la recursividad como una herramienta para mejorar su desempeño en la construcción de programas que den solución efectiva a problemas específicos en el campo del desarrollo de software. Este proyecto fue financiado por la Red de Universidades Regionales Latinoamericanas -UREL - capítulo Nariño. Este libro es uno de los resultados de esta investigación.

Este libro se elaboró como texto guía para los estudiantes de las diferentes instituciones que participaron en este proyecto y está pensado como un texto guía de un curso introductorio de programación, para estudiantes sin conocimientos previos sobre programación. El contenido de este libro está diseñado para ser cubierto en un semestre académico y puede ser seguido de una asignatura de programación imperativa.

La organización de los capítulos llevará al lector a conocer desde los conceptos a las diferentes estructuras de programación, y todos los aspectos del lenguaje Scheme se estudian progresivamente con aplicaciones prácticas en Dr. Scheme.

El *capítulo 1* es una introducción a los diferentes paradigmas de programación, haciendo énfasis en el paradigma funcional.

El *capítulo 2* describe los fundamentos del lenguaje SCHEME, su historia, sus características, elementos y la estructura de un programa en SCHEME.

El *capítulo 3* trata las sentencias básicas de entrada y salida con SCHEME.

El *capítulo 4* aborda los diferentes tipos de estructuras condicionales que maneja el lenguaje SCHEME.

El *capítulo 5* estudia las estructuras repetitivas, comúnmente conocidas como ciclos y su forma de implementarlos con el lenguaje SCHEME.

El *capítulo 6* explica la recursividad como técnica de programación y la estructura y ejecución de funciones recursivas en SCHEME.

El *capítulo 7* aborda el manejo de arreglos, su definición y las funciones de SCHEME para su manipulación.

El *capítulo 8* termina el libro con el estudio de listas, su definición y manejo con el lenguaje SCHEME.

En todos los capítulos se incluyen varios ejemplos que permiten al lector un mayor entendimiento de los temas. Para facilitar la ubicación del código fuente, en el apéndice A se presenta una lista de ejemplos.

Los programas de ejemplo, el código fuente y documentación adicional sobre este tema puede consultarse en el sitio web:

http://virtual.iucesmag.edu.co/ruisnar/scheme

Agradecimientos

A la Red de Universidades Regionales Latinoamericanas UREL, capítulo Nariño por hacer posible la realización del proyecto de investigación y la publicación de este libro.

A Gonzalo Hernández, quien participó en las dos primeras fases de la investigación y colaboró en el estudio del lenguaje Scheme y en la capacitación de los investigadores de RUISNAR.

A Wilson Pantoja, quien participó en los primeros meses de investigación en representación de la Fundación Universitaria San Martín y colaboró en el estudio del editor PLT Dr. Scheme.

A Johana Muñoz, quien participó en la primera fase de investigación y colaboró en la revisión bibliográfica sobre los fundamentos del lenguaje Scheme.

A Constanza Colunge, quien participó en la tercera fase de la investigación en representación de la Corporación Universitaria Autónoma de Nariño.

A Mirian Benavides, Sonia Narváez y José María Muñoz por el tiempo que dedicaron a revisar el documento y por sus acertadas observaciones.

CAPÍTULO 1

PARADIGMAS DE PROGRAMACIÓN

Tan solo por la educación
puede el hombre llegar a ser hombre.
El hombre no es más que
lo que la educación hace de él.
Kant

Un paradigma está constituido por los supuestos teóricos generales, las leyes y las técnicas para su aplicación que adoptan los miembros de una determinada comunidad científica. Es un marco de referencia que impone reglas sobre cómo se deben hacer las cosas, indica qué es válido dentro de éste y qué está fuera de sus límites. Un paradigma distinto implica nuevas reglas, elementos, límites y maneras de pensar, o sea implica un cambio. Los paradigmas pueden ser considerados como patrones de pensamiento para la resolución de problemas.

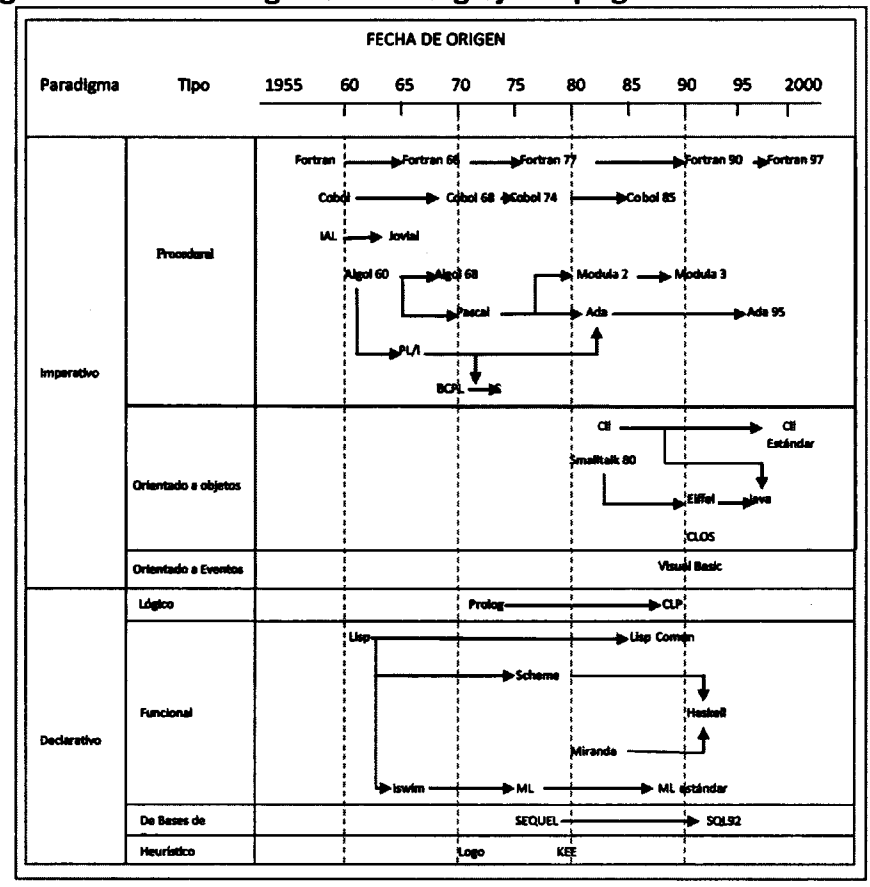
1.1 EL CONCEPTO DE PARADIGMA DE PROGRAMACIÓN

Un paradigma de programación es una colección de modelos conceptuales que orientan el proceso de diseño y desarrollo de programas con unas directrices específicas, tales como: estructura modular, fuerte cohesión y alta rentabilidad. La estructura conceptual de dichos modelos está pensada de modo tal que determinan la forma correcta de los programas y esquematizan la manera de pensar y formular soluciones que deben expresarse mediante un lenguaje de programación.

Los paradigmas de programación representan un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro; sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro.

Un paradigma de programación esta asociado con cierta familia de lenguajes de programación (ver figura 1.1).

Figura 1.1. Historia de algunos de los lenguajes de programación



Muchos lenguajes se diseñan para poner un paradigma en ejecución, por ejemplo, Smalltalk y Java se asocian a la Programación Orientada a Objetos, mientras que Scheme y Haskell se asocian a la Programación Funcional. Otros lenguajes, tales como Common Lisp, Python, y Oz son para permitir el uso de paradigmas múltiples. (GNU Free Documentation License).

El desarrollo de los lenguajes de programación, sigue de cerca el desarrollo de los procesos físicos y electrónicos usados en las computadoras, muchos han tenido una vida y utilidad limitada, mientras que otros han tenido un éxito en su aplicación en uno o más dominios hasta el punto que han influído en el diseño de los lenguajes futuros. En la figura 1.1 se describe el resumen general de la historia de algunos de los lenguajes de programación que han tenido más influencia (Tucker y Noonann, 2003).

1.2 TIPOS DE PARADIGMAS DE PROGRAMACIÓN

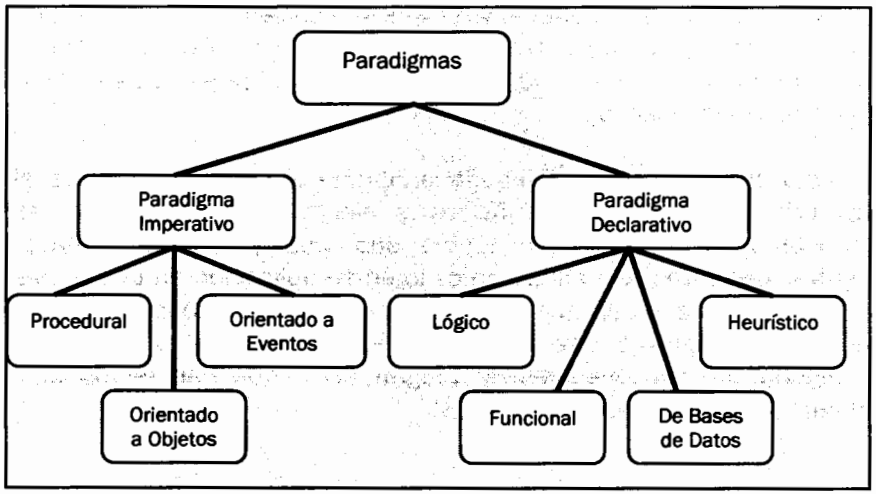
Dado que un paradigma es un conjunto de principios generales que un programador ha de seguir para desarrollar un programa, cada lenguaje de programación permite al programador seguir un paradigma determinado, por tanto los lenguajes de programación se pueden también clasificar según el paradigma que siguen.

Una clasificación básica de los paradigmas de programación se muestra en la figura 1.2.

1.2.1 Paradigma Imperativo

La *programación imperativa* recibe este nombre debido a que está basada en comandos que actúan sobre variables. Este paradigma es el más antiguo (1950s), los primeros diseñadores se dieron cuenta que la utilización del concepto de variable unido al comando de asignación constituían una abstracción simple pero muy efectiva para poder controlar la información almacenada en la memoria de un computador. (Serón, Magallón y Baldassarri, 2006)

Figura 1.2. Clasificación básica de los paradigmas de programación



Se determina también como un modelo abstracto que consiste en un gran almacenamiento de memoria donde la computadora guarda una representación codificada de un cálculo y ejecuta una secuencia de comandos que modifican el contenido de ese almacenamiento.

Un programa en un lenguaje imperativo aparece como una lista de instrucciones u órdenes elementales que han de ejecutarse una tras otra, en el orden en que aparecen en el programa. Las instrucciones de un programa imperativo utilizan datos almacenados en la memoria del computador, llamados variables. Para realizar algún cálculo, se parte de ciertos datos almacenados y se realizan diversas operaciones (instrucciones); al final, el resultado está almacenado en alguna celda de memoria.

A este tipo de paradigma de programación se le suele llamar algorítmico, dado que el significado de algoritmo es análogo al de receta, método, técnica, procedimiento o rutina, y se define como un conjunto finito de reglas diseñadas para crear una secuencia de operaciones para resolver un tipo específico de problemas. De esta forma para Wirth, un programa viene definido por la ecuación

Programas = Algoritmos + Estructura de Datos

Los elementos principales de la programación imperativa son:

- Variable. El componente principal es la memoria, compuesto por un gran número de celdas donde se almacenan los datos y que son referenciadas por medio de su nombre (variable). El conjunto de valores de todas las variables del programa en un momento dado representa el estado del programa.

- Operaciones de asignación. Cada valor calculado debe ser asignado a la variable mediante operaciones de asignación. De esta forma se modifica el estado del programa.

- Bifurcación. Es una estructura base de decisión en un lenguaje imperativo.

- Repetición. Un programa imperativo, normalmente realiza su tarea ejecutando repetidamente una secuencia de pasos elementales.

El paradigma imperativo es una abstracción del lenguaje ensamblador. Algunos de los lenguajes de programación que siguen el paradigma imperativo son: Basic, Pascal, Modula y C.

Una clasificación importante del paradigma imperativo es:

Procedural (estructurado). Es un paradigma de programación basado en el concepto de llamado de procedimientos, también conocidos como rutinas, subrutinas, métodos o funciones simplemente contienen series de pasos computacionales, cualquier procedimiento puede ser llamado en cualquier punto durante la ejecución de un programa, incluyendo otros procedimientos o en él mismo. Algunos lenguajes de programación procedimental son: Pascal, C, Basic

Paradigma orientado a objetos. Es un paradigma de programación que define los programas en términos de "clases de objetos", los objetos son entidades que combinan *estado* (es decir, datos) y *comporta-*

miento (esto es, procedimientos o métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas. Esto difiere de los lenguajes procedurales tradicionales, en los que los datos y los procedimientos están separados y sin relación, lo que no sucede con los lenguajes orientados a objetos donde los datos y los procedimientos (métodos) forman una entidad llamada *objeto*. Estos métodos están pensados para hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

Una ecuación para reconocer una aproximación a la Orientación a Objetos es (Jimenez, 2005):

Orientación_Objetos = Clases + Objetos + Herencia + Comunicación con mensajes

Los elementos principales de la programación orientada a objetos son

- Objetos (Sinónimo de instancia). Es la abstracción de alguna cosa en el dominio del problema que refleja la capacidad de un sistema de alcanzar información alrededor de él. Los objetos, por lo tanto, son entidades que tienen atributos (datos) y formas de comportamiento (procedimientos) particulares.

- Clase. Una clase es una descripción de un conjunto de objetos casi idénticos. Una clase consta de métodos y datos que resumen las características comunes de los objetos, incluyendo una descripción de cómo crear un nuevo objeto de la clase. En otras palabras, las clases contienen los anteproyectos para crear objetos.

- Herencia. Es un mecanismo para expresar similaridad entre clases, simplificando definiciones de las clases similares previamente definidas. La herencia permite crear nuevas clases llamadas subclases agregando solamente las diferencias con la clase. En otras palabras la herencia es una partición en subclases más especializadas.

- Comunicación con mensajes. Los objetos tienen la posibilidad de actuar, la acción sucede cuando un objeto recibe un mensaje, que es, una solicitud que pide al objeto que se comporte de alguna forma. Cuando se ejecutan los programas orientados a objetos, los objetos reciben, interpretan y responden a mensajes procedentes de otros objetos. Los mensajes pueden contener información para clasificar una solicitud. Los procedimientos residen en el objeto y determinan cómo actúa el objeto cuando recibe un mensaje. De hecho, los métodos proporcionan el único mecanismo para cambiar los valores de las variables del objeto. (A este hecho se le conoce como encapsulamiento). Los mensajes que reciben el objeto son los únicos conductos que conectan al objeto con el mundo exterior. Estas características de los objetos confieren a la orientación a objetos su ventaja: la orientación a objetos fomenta la modularidad haciendo muy claras las fronteras entre objetos, explícita la comunicación entre los mismo y oculta los detalles de la realización.

A este paradigma pertenecen los lenguajes de programación: SmallTalk, Java, C++.

Paradigmas orientados a eventos. Es un paradigma de programación que se centra en la definición de acciones sobre objetos predefinidos, denominadas ***eventos***. Son eventos típicos: el click sobre un objeto de tipo botón, el hacer doble click sobre el nombre de un archivo para abrirlo, el arrastrar un icono, el pulsar una tecla o combinación de teclas, el elegir una opción de un menú, el escribir en una caja de texto, o simplemente mover el ratón entre otros.

A este paradigma pertenecen los lenguajes de programación: Visual Basic, C Builder, Kylix, Power Builder.

1.2.2 Paradigma Declarativo

El paradigma declarativo señala las características que debe tener la solución, sin describir cómo procesarla. Es decir, especifica qué se desea obtener pero no cómo obtenerlo. Un programa declarativo se construye señalando hechos, reglas, restricciones, ecuaciones,

transformaciones y otras propiedades derivadas del conjunto de valores que configuran la solución. La programación en un lenguaje declarativo es potencialmente de más alto nivel que la programación imperativa.

La característica fundamental del paradigma declarativo es que no existe la asignación ni el cambio de estado en un programa. Las variables son identificadores de valores que no cambian en toda la evaluación. Sólo existen valores y expresiones matemáticas que devuelven nuevos valores a partir de los declarados. A partir de esta información, el sistema debe de ser capaz de derivar un esquema de evaluación que permita computar una solución, es decir, no existe una descripción paso a paso de cómo llegar a la solución.

En términos generales, al desarrollar software en el paradigma de programación declarativo, en contraste con el Imperativo, se tiende a enfatizar más la evaluación de expresiones que la ejecución secuencial de comandos.

Las soluciones declarativas son más cercanas a la manera en que funciona la mente humana al permitir a los programadores describirlas como sistema de expresiones que serán evaluadas. Por otro lado, las soluciones imperativas, al forzar la formulación de algoritmos como una serie de pasos que se realizarán de manera secuencial, son semejantes a la manera en que funciona el computador.

Una clasificación importante del paradigma declarativo es:

Paradigma Lógico. La programación lógica está basada en el cálculo de predicados, incluyendo instrucciones escritas en formas conocidas como cláusulas de Horn. El cálculo de predicados proporciona axiomas y reglas de modo que se puede deducir nuevos hechos a partir de otros ya conocidos. Una cláusula de Horn permite que sólo un nuevo hecho sea deducido en cualquier instrucción simple. Un sistema de cláusulas de Horn permite un método particularmente mecánico de demostración llamado *resolución* (Appleby y Vandekopple, 1998).

Un programa basado en lógica se compone de una serie de axiomas o hechos, reglas de inferencia y un teorema o cuestión por demostrarse. La salida es verdadera si los hechos soportan o apoyan la cuestión, y es falsa en el caso contrario.

A este paradigma pertenecen los lenguajes de programación: Prolog y CLP.

Paradigma Funcional. El objetivo del paradigma funcional es conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, evitando el concepto de estado del cómputo. La secuencia de computaciones llevadas a cabo por el programa se rige única y exclusivamente por la reescritura de definiciones más amplias a otras cada vez más concretas y definidas, usando lo que se denominan *definiciones dirigidas* (Guzman, 2000).

Una *definición dirigida* es un modelo matemático de composición funcional donde el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produce el valor deseado.

En el paradigma Funcional no existe el concepto de celda de memoria que es asignada o modificada; más bien, existen valores intermedios que son el resultado de cálculos anteriores y las entradas a cálculos subsiguientes, ni tampoco existen sentencias imperativas. La programación funcional incorpora el concepto de función como objeto de primera clase, lo que significa que las funciones se pueden tratar como datos (pueden pasar como parámetros, calculadas y devueltas como valores normales, y mezcladas en el cálculo con otras formas de datos).

En este paradigma se concibe la solución como una composición de funciones. Por ejemplo, para ordenar una lista, se puede diseñar la solución como una concatenación de listas más pequeñas, cada una de las cuales ya está clasificada. Esto reduce el problema a seleccionar las listas más pequeñas.

A este paradigma pertenecen los lenguajes de programación: Scheme, Commonlisp.

Paradigmas del lenguaje de base de datos. Las propiedades que distinguen a los lenguajes diseñados para tratar con bases de datos son la persistencia y la administración de cambios. Las entidades de bases de datos no desaparecen después de que finaliza un programa, sino que permanecen activas durante tiempo indefinido como fueron estructuradas originalmente. Puesto que la base de datos, una vez organizada, es permanente, estos lenguajes también deben soportar los cambios. Los datos pueden cambiar y así también pueden hacerlo las relaciones entre objetos o entidades de datos (Appleby y Vandekopple, 1998).

Un sistema de administración de base de datos incluye un *lenguaje de definición de datos* (DDL) para describir una nueva colección de hechos, o datos, y un *lenguaje de manipulación de datos* (DML) para la interacción con las bases de datos existentes.

A este paradigma pertenecen los lenguajes de programación: Sequel, Sql.

Paradigma Heurístico. Se puede definir como aquel tipo de programación computacional que aplica para la resolución de problemas reglas de buena lógica (reglas del pulgar) denominadas heurísticas, las cuales proporcionan entre varios cursos de acción uno que presenta visos de ser el más prometedor, pero no garantiza necesariamente el curso de acción más efectivo (Barzanallana, 2005).

Una heurística es la conclusión del razonamiento humano en un dominio específico. Por ello, este tipo de programación se aplica con mayor intensidad en el campo de la Inteligencia Artificial (IA), y en especial, en el de la Ingeniería del Conocimiento, dado que el ser humano opera la mayoría de las veces utilizando heurísticas.

La Programación Heurística implica la aplicación del conocimiento específico integral de la naturaleza del problema a resolver, que

comprende, además de su operatoria, la manera cómo se genera y que efectos produce. Es una forma de modelar el problema en lo que respecta a la representación de su estructura, estrategias de búsqueda y métodos de resolución. Esto permite minimizar el tiempo de búsqueda de las alternativas más eficaces.

Este paradigma de programación se presenta y utiliza desde diferentes puntos de vista:

Como técnica de búsqueda para la obtención de metas en problemas no algorítmicos, o con algoritmos que generan explosión combinatoria, por ejemplo juegos como damas y ajedrez.

Como un método aproximado de resolución de problemas utilizando funciones de evaluación de tipo heurístico por ejemplo algoritmos A* y AO*

Como método de poda para estrategias de programas que juegan, aunque estos métodos no son realmente heurísticos por ejemplo poda alfa-beta.

La Programación Heurística no ha producido un lenguaje específico de programación, debido a que las heurísticas, al *ser reglas de sentido común*, se pueden implementar con cualquiera de los lenguajes descritos en los diferentes paradigmas de programación. Por otra parte, al aplicarse este tipo de programación, fundamentalmente, en el campo de la I.A., las heurísticas están siendo implementadas mayoritariamente en herramientas de esta área.

A este paradigma pertenecen los lenguajes de programación: Logo, Prolog, Kee entre otros.

1.3 PARADIGMA FUNCIONAL

1.3.1 Historia

Está basado completamente en un enfoque funcional, en la creación y uso de funciones matemáticas. Este modelo computacional estableció las bases teóricas de lo que más tarde serían los lenguajes de programación funcional.

La popularidad del paradigma funcional se debe en gran medida al éxito del lenguaje de programación Lisp y de sus dialectos, como Scheme. Tanto el Lisp como Scheme son lenguajes que se están utilizando en la actualidad.

Los orígenes teóricos del modelo funcional se remontan al año 1934, cuando Alonso Church introdujo un modelo matemático de computación llamado *lambda cálculo*.

El Cálculo Lambda es una forma simple de modelar los aspectos computacionales de las funciones. Su estudio ayuda a comprender los elementos de la programación funcional y de la semántica que subyace en los lenguajes, independientemente de los detalles sintácticos de un lenguaje particular de programación funcional. Su nombre proviene de la letra griega "ƛ" (Lambda) que se utiliza en la notación (Seron, Magallón y Baldassarri, 2006).

A pesar de que en 1934 las computadoras aun no existían, el cálculo lambda se puede considerar como el primer lenguaje funcional de la historia y sus fundamentos fueron la base de toda la teoría de la programación funcional y de los lenguajes funcionales desarrollados posteriormente. Se puede decir que los lenguajes funcionales modernos son versiones de lambda cálculo con numerosas ayudas sintácticas.

1.3.2 Programación funcional

El uso de funciones, con comportamiento análogo al de las funciones matemáticas, para crear programas es lo que se denomina "programación funcional". Un ingrediente esencial de la programación funcional es que el valor de una función queda determinado exclusivamente por el valor de sus argumentos. Por lo tanto cualquier llamada a la función con los mismos argumentos producirá siempre el mismo valor. En programación funcional las funciones deben cumplir con la propiedad de transparencia referencial.

El objetivo del diseño de la programación funcional es intentar copiar el funcionamiento de las funciones matemáticas de la manera más extensa posible. Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la transparencia referencial (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de efectos laterales.

1.3.3 Lenguajes Funcionales

Los matemáticos han resuelto problemas usando el concepto de función, la cual convierte ciertos datos en resultados. Si se conoce cómo evaluar una función usando la computadora, se pueden resolver automáticamente muchos problemas. Este análisis fue realizado por algunos matemáticos que consideraron la computadora una herramienta importante para su trabajo, creando los lenguajes de programación funcionales, los que aprovechan la posibilidad que tienen las funciones para manipular datos simbólicos, y no solamente numéricos, y la propiedad 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 (Steele y Sussman, 1979).

Un lenguaje funcional ofrece conceptos que son muy entendibles y relativamente fáciles de manejar. El lenguaje funcional más antiguo, y seguramente el más popular hasta la fecha, es LISP, diseñado por McCarthy en la segunda mitad de los años 50. Su área de aplicación es principalmente la Inteligencia Artificial. En la década de los 80 hubo una nueva ola de interés por los lenguajes funcionales, añadiendo la tipificación y algunos conceptos modernos de modularización y polimorfismo. (Mcarthy, 1965)

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.

Un lenguaje de programación funcional proporciona:

- Un conjunto de funciones primitivas
- Un conjunto de formas funcionales para construir funciones complejas.
- Una operación de aplicación de las funciones
- Algunas estructuras para almacenar datos

La ejecución de un lenguaje de programación funcional está basada en dos mecanismos fundamentales:

- Asociación de los valores con los nombres (argumentos de las funciones).
- Aplicación de las funciones para calcular nuevos valores

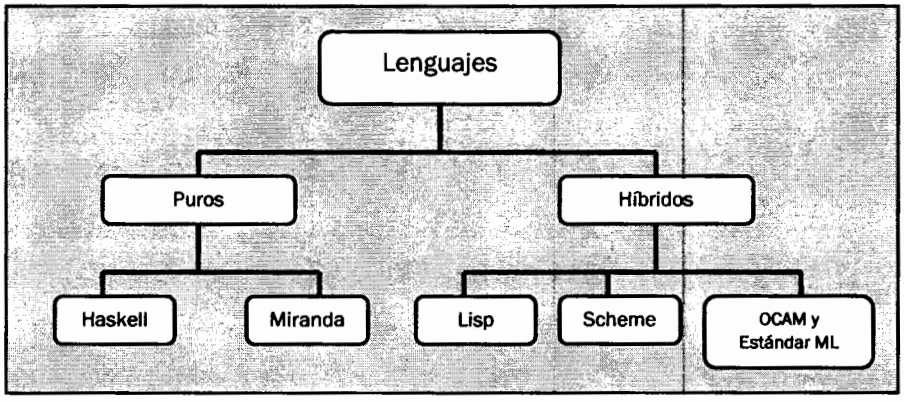
Un lenguaje de programación funcional no necesariamente utiliza variables con el sentido de localización de memoria, ni requiere por lo tanto sentencias de asignación con el significado de los lenguajes imperativos ni la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas), lo que permite liberar al programador del control de las celdas de memoria.

Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes procedimentales, como las secuencias de instrucciones o la asignación de variables.

En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje funcional híbrido.

Entre los lenguajes funcionales puros, cabe destacar a Haskell y Miranda. Los lenguajes funcionales híbridos más conocidos son: Lisp, Scheme, Ocaml y Standard ML (estos dos últimos, descendientes del lenguaje ML). Ver figura 1.3

Figura 1.3 Categoría de los lenguajes funcionales



Las características fundamentales de los lenguajes funcionales de programación son, según (Guzman, 2000):

El valor de una expresión depende sólo de los valores de sus subexpresiones, si las tiene. La programación funcional pura es una programación sin asignaciones. En realidad, la mayoría de los lenguajes funcionales son impuros, ya que permiten asignaciones. Sin

embargo, su estilo de programación es diferente al de los lenguajes de programación imperativa.

Almacenamiento implícito. El programador no debe preocuparse en manejar el almacenamiento de datos. Una consecuencia de esto es que la implementación del lenguaje debe realizar una "recolección de basura" para recuperar la memoria que se ha usado y no se volverá a utilizar.

Las funciones son valores de primera clase. Una función puede ser el valor de una expresión, pasarse como argumento o colocarse en una estructura de datos. Esto permite potentes operaciones.