





**Faster, Safer, Smarter browser**

  Free VPN, Ad blocker, Battery Saver, Video Popout, News Reader, Cu

**Faster, Safer, Smarter browser**

  Free VPN, Ad blocker, Battery Saver, Video Popout, News Reader, Cu

mucho mas los conceptos de programación al mundo real y fue allí en donde se encontró que todo lo que nos rodea tiene unas características y sirve para algo. Por ejemplo un lápiz tiene peso, color, olor, sabor (si se quiere), longitud, espesor, torque, textura y muchas otras características. Al mismo tiempo un lápiz sirve para escribir, para separar una hoja de un libro, para rascarse la espalda, para defenderse de un atraco, para señalar un punto, para dibujar, para manchar a alguien y para miles de cosas mas.

Esta concepción llevó a una gran revolución en la historia de la programación pues se crearon dos vertientes dentro de la lógica de programación: La programación estructurada que ya definimos y la *Programación Orientada A Objetos* por medio de la cual se podía modelar el mundo en el computador tal y como es. Su aporte principal era el concepto de objeto. Qué es pues un objeto..? En términos generales un objeto no es mas que un ente informático que tiene características (técnicamente llamadas atributos) y que sirve para algo (técnicamente se dice que tiene unos métodos).

Así se creó pues este concepto y se comenzaría a utilizar los objetos (en programación) que como ya dijimos no son mas que tipos de datos con atributos y métodos propios. Toda una teoría se comenzó a derivar de esta nueva concepción del mundo y se fue aplicando poco a poco en la programación ya que se empezaron a descubrir unas relaciones entre objetos, unas operaciones entre objetos y en general un montón de conceptos nuevos en cuanto a lo que inicialmente no habían sido mas que los objetos. Mientras se desarrollaba esta teoría y se ponía en práctica en muchos de los Lenguajes de Programación comerciales también se seguía utilizando la técnica de programación estructurada pues estas dos técnicas no eran excluyentes. Dependía pues del programador que tuviera una verdadera concepción acerca del problema que quería solucionar la decisión de saber por cuál técnica de programación (programación estructurada o programación orientada a objetos) era mas apropiado resolverlo.

Ello exigía simultáneamente que el programador no solo conociera muy bien los conceptos de programación sino que también conociera muy bien el problema que iba a solucionar y las características de cada una de las técnicas de programación. Tenía que ser un profesional integral de la programación pues ahora no solo se necesitaba que supiera de computadores o de electrónica o que se supiera las instrucciones de un simple lenguaje. Ahora tenía que conocer teoría y combinarlas de manera que pudiera llegar a la mejor solución aprovechando la tecnología existente. Debo decir que cuando comenzó a tomar fuerza la teoría de la programación orientada a objetos no todo lo Lenguajes de Programación (o mas bien no todos sus compiladores) estaban acondicionados para que aceptaran la nueva forma de programar.

De esta manera también era necesario que el programador supiera si el problema que iba a solucionar a través de un programa era implementable fácilmente con el Lenguaje de Programación que tuviera a la mano pues debe usted saber que no es fácil inducir la compra de un Lenguaje de Programación (o sea de su compilador) en una empresa cuando todo el sistema de información está basado en otro Lenguaje de Programación. Esta filosofía de programación fue cogiendo mucha fuerza y con ella se fueron fortaleciendo los lenguajes que habían iniciado la aceptación de esas nuevas características. Empresas fabricantes que hasta ese momento habían sido competitivas se convirtieron en verdaderos imperios de la informática. La programación definitivamente había dado un salto impresionante hacia la solución de muchos problemas que eran, en algunos casos, mas complejos de resolver con programación estructurada que con programación orientada a objetos.

Poco a poco algunos fabricantes de Lenguajes de Programación se fueron introduciendo en el mercado y aprovechando las características de la nueva técnica de programación fueron dejando de lado, de alguna manera, la programación estructurada que algunos libros han llamado erróneamente Programación Tradicional. En ese avance tecnológico y con el ánimo de entregar al mercado de la programación mas y mejores herramientas de trabajo se empezó a manejar un concepto muy importante en programación como es el concepto de interfaz. Una interfaz no es mas que la forma como usted puede mostrar la información por medio de algún dispositivo de salida. Ya se sabía que entre mas clara y entendible fuera la información podría decirse que los programas serían mejores ya que lo que finalmente el usuario de un programa necesitaba era que la información que le arrojaba un computador fuera claramente entendible.

Se fue notando pues, por parte de las empresas fabricantes de Lenguajes de computadores, como el tiempo de un programador se iba en su mayor parte en el diseño de las interfaces o sea en el diseño de la presentación de los datos. Por tal motivo se pensó que, en unión con la teoría de programación orientada a objetos y con las herramientas que ella facilitaba, se hacía necesario diseñar lenguajes de programación que facilitaran el diseño de interfaces para que el programador invirtiera su tiempo mejor en el diseño de procesos o de manipulación y tratamiento de datos.

Fue entonces cuando entraron al mercado los Lenguajes Visuales y se incorporó al desarrollo de la programación la *Programación Visual* que no es mas que una forma de programar en donde se cuenta con una gran cantidad de herramientas prediseñadas para facilitar, precisamente, el diseño de interfaces. Este tipo de programación ha llevado a que en el mundo de la informática y exactamente en la programación se llegue a unos resultados mucho mas convenientes y mejores a nivel técnico pues en la actualidad se pueden obtener aplicaciones de computador mucho mas entendibles y manejables por el usuario gracias a la filosofía incorporada por la Programación Visual.

A este nivel la programación requería menos conceptos técnicos y mas lógica de programación que era lo que realmente se necesitaba para desarrollar un programa. Es normal ver como una persona con unos modestos conocimientos de computación puede, a través de Lenguajes Visuales, desarrollar aplicaciones verdaderamente útiles y además muy bien presentadas. Lo que poco a poco se fue afianzando fue la necesidad de tener unos conceptos de lógica de programación bien fundamentados para poder aprovechar de una manera eficiente los recursos que la informática le entregaba al computador.

Como se busca modelar con el computador al mundo que nos rodea y en ese avance la tecnología cada vez se ha ido mejorando mas y mas se espera que dentro de muy poco se podrá hablar de una *Programación Virtual* en donde el programador pueda ver en tres dimensiones (3D) todo el escenario que necesita para crear sus aplicaciones. Es muy posible que cuando este libro esté en sus manos algunos de estos lenguajes de programación ya estén en el mercado.

Tenga en cuenta que en un país como el nuestro (dependiente de la tecnología que viene del exterior) los avances tecnológicos van un poquito rezagados comparando con los países industrializados dado que es allá en donde se desarrolla la llamada Tecnología de Punta que no son mas que los nuevos avances tecnológicos que le permiten al hombre tener mas y mejores herramientas para aplicarlas en la solución de sus necesidades (y volverse dependiente de ellas).

## Capítulo 6

---

# *Metodología, Técnica Y Tecnología Para Solucionar Un Problema*

Hasta este momento tenemos una metodología para solucionar un problema, conocemos unas técnicas para representar la solución y hemos hablado de la tecnología a nivel de lenguajes de programación para que el computador cumpla por nosotros el objetivo propuesto. Todo esto se une en una teoría que nos permite acercarnos a la lógica de programación y por supuesto, gracias al primer contacto que ya tenemos con los lenguajes, a la programación como tal. Es importante saber que cuando se habla de lógica de programación se está hablando de ese conjunto de normas técnicas que nos permiten que de una manera sencilla nosotros desarrollemos un algoritmo entendible para la solución de un problema y cuando se habla de programación como tal es la utilización de lenguajes que permiten que nuestra solución sea entendida y ejecutada por un computador.

Precisamente y con el ánimo de ilustrar todo la teoría que hasta el momento hemos visto, vamos a dar tres enunciados y vamos a resolverlos aplicando toda la metodología para solucionar un problema, utilizando las técnicas de representación y codificándolos en unos lenguajes de programación. Para ello vamos a tener en cuenta que la teoría expuesta hasta el momento se resume en los siguientes tópicos:

### 1. Concepción del problema

---

Es muy importante que cuando tengamos un enunciado podamos tener de él una concepción acertada de manera que nos podamos alcanzar un objetivo y que ése objetivo sea el que

realmente necesita ser solucionado. La concepción del problema es el camino para tener la certeza de que lo hemos entendido correctamente y que lo que buscamos solucionar coincide con lo que se busca solucionar en el problema. Dentro de la concepción del problema tendremos las siguientes etapas:

**a. Clarificación del objetivo**

Por lo dicho en capítulos anteriores es muy importante que a través de un razonamiento teórico y textual nos sentemos a reflexionar en cuanto a los alcances de nuestro objetivo (enunciado como un problema) ya que con eso tendremos muy claro no solo hacia donde debemos ir sino hasta donde debemos llegar.

**b. Algoritmo**

Es el conjunto de pasos que nos permiten llegar (ojalá de la mejor de las formas) a alcanzar el objetivo propuesto. Debe ser organizado y, ante todo, ordenado para que sea absolutamente entendible.

**c. Prueba de Escritorio**

Es la prueba reina de un algoritmo. Nos permite saber si realmente está bien o no. Cuando un algoritmo está bien...? Solamente cuando realmente alcanza el objetivo propuesto. Si un algoritmo no alcanza el objetivo que inicialmente se propuso estará mal así haga maravillas en su desarrollo.

## 2. Técnicas de Representación

---

Es importante que usted conozca y domine las técnicas de representación porque con ello usted podrá evaluar ventajas y desventajas reales (y para usted) y podrá determinar cuál es la técnica mas apropiada para la representación de sus algoritmos. No está de mas decir que cuando se quiere representar un algoritmo solamente se utiliza una de las técnicas pero para los objetivos explicativos de este libro representaremos los algoritmos de este capítulo y de otros subsiguientes con las tres técnicas, solo para que usted encuentre diferencias entre ellos y ello le permita realizar una correcta evaluación y establecer unos criterios firmes acerca de su utilización.

**a. Diagramas de Flujo**

Representados por signos en donde el hilo conductor de la lógica se representa por flechas que van a significar la dirección del flujo de la idea.

#### **b. Diagramación Rectangular Estructurada**

Esquema en donde se utiliza un rectángulo como base y utilizando solo tres tipos de notaciones se puede representar todo lo que para nosotros sea parte de un algoritmo.

#### **c. Seudocódigo**

Texto basado en unas normas técnicas que lo hace muy entendible y sobre todo muy fácil de codificar y que representa, obviamente, la solución que hayamos planteado a través de un algoritmo.

### **3. Transcripción o Codificación**

---

Es la representación de un algoritmo a través de un Lenguaje de Programación. En este capítulo utilizaremos los lenguajes *Basic*, *Pascal*, *C* y *Cobol* como ejemplos y explicaremos brevemente y de manera muy somera, ya que no es el objetivo del libro, algunos tópicos acerca de cada uno de los lenguajes. También es importante que usted sepa que cuando vaya a desarrollar realmente programas aplicativos solo va a tener que codificar en un solo lenguaje de programación. En este libro lo haremos en cuatro lenguajes solo por el ánimo explicativo del libro y para establecer algunas diferencias entre uno y otro lenguaje.

## **PRIMER ENUNCIADO**

Desarrollar un programa que permite leer un número entero positivo y determinar si es par.

### **1. Concepción del problema**

---

#### **a. Clarificación del objetivo**

Se trata de recibir un número entero (para lo cual utilizaremos una variable de tipo entero), verificar que es un número positivo y determinar si es un número par. Recordemos pues que son números pares aquellos que son divisibles exactamente entre dos, o sea, aquellos que al dividirlos entre 2 su residuo es cero. Algunos números pares son 18, 6, 4, 56 y 88. Algunos números que no son pares son 45, 7, 19, 23 y 99 ya que no cumplen con las condiciones de los números pares. En caso de que el número leído sea para avisaremos a través de un título que el *número sí es par* y en caso de que no sea así entonces haremos lo mismo avisando que el *número no es par*. Apenas hayamos avisado a través de un título que el número es par o que no lo es, entonces allí deberá terminar nuestro algoritmo.

**b. Algoritmo**

*Algoritmo para determinar si un número es par*

*Inicio*

*Leer un número y guardarlo en una variable entera*

*Si ese número es negativo*

*Escribir que ese número no sirve para nuestro propósito*

*Sino*

*Preguntar si el número es par*

*Si lo es entonces escribir que el número leído es par*

*Si no lo es escribir que el número leído no es par*

*Fin*

Ya de por sí debemos tener en cuenta que el algoritmo es en sí la esencia de nuestra idea tal y como está representado aquí. Ahora lo que tenemos que hacer es ir mutando nuestra idea para que se convierta en una solución mas aproximada a lo técnico que a lo informal. Comencemos pues con un análisis detallado de cada una de las órdenes que aparecen en este algoritmo:

Si vamos a convertir este algoritmo en un programa entonces el nombre debe ser un poco mas técnico. De esta manera no lo vamos a llamar *Algoritmo para determinar si un número es par* sino que lo vamos a llamar *Algoritmo Número\_Par* y será nuestra obligación recordar que el Algoritmo Número\_Par es el algoritmo que nos permite leer un número y determinar si es par.

Como vamos a tener la necesidad de utilizar una variable para que almacene el número que se va a leer entonces es necesario que al inicio del algoritmo declaremos una variable de tipo entero a la cual vamos a llamar (por conveniencia técnica) num, de esta forma la cabecera de nuestro algoritmo que estaba así

*Algoritmo para determinar si un número es par*

*Inicio*

Se va a transformar en

*Algoritmo Número\_Par*

*Variables*

*Entero: num*

*Inicio*

Esto significa que durante el algoritmo vamos a utilizar una variable que la vamos a llamar num, que solo podrá almacenar datos de tipo entero y que cuando se utilice en operaciones sus resultados se van a regir por las reglas de la aritmética entera (es decir sin decimales).

Como ya tenemos una variable en donde vamos a almacenar el número que se lea entonces la orden *Leer un número y guardarlo en una variable entera* se convertirá conceptualmente en *Leer*

un número y guardarlo en la variable entera *num* que por lo dicho en capítulos anteriores es lo mismo que decir *Lea num* (orden dada al computador).

Entonces nuestro algoritmo que hasta el momento era

*Algoritmo para determinar si un número es par*

*Inicio*

*Leer un número y guardarlo en una variable entera*

Se convierte ahora en

*Algoritmo Número\_Par*

*Variables*

*Entero : num*

*Inicio*

*Lea num*

Como ya tenemos el valor guardado en una variable entonces preguntar por el número leído es lo mismo que preguntar por el contenido de la variable y hemos de recordar que cuando se utiliza el nombre de una variable en un algoritmo eso representará que nos estamos refiriendo al contenido de dicha variable. Igualmente preguntar si un número es negativo se reduce a preguntar si dicho número es menor que 0 valiéndonos de un operador relacional ( $<$ ). En esas condiciones la pregunta

*Si ese número es negativo*

*Escribir que ese número no sirve para nuestro propósito*

Se convierte en

*Si num < 0*

*Escriba "El número debe ser positivo"*

Y por lo tanto nuestro algoritmo que originalmente era

*Algoritmo para determinar si un número es par*

*Inicio*

*Leer un número y guardarlo en una variable entera*

*Si ese número es negativo*

*Escribir que ese número no sirve para nuestro propósito*

Se convierte ha transformado, hasta el momento, en

*Algoritmo Número\_Par*

*Variables*



```

Entero : num
Inicio
  Lea num
  Si num < 0
    Escriba "El número debe ser positivo"

  Sino
    Preguntar si el número es par
    Si lo es entonces escribir que el número leído es par
    Si no lo es escribir que el número leído no es par
Fin

```

Que consistirá en determinar si el número es par para avisar a través de un título que Sí lo es o que No lo es. Pero como convertimos técnicamente la pregunta *Si el número es par* para que el computador la pueda ejecutar y obtener la respuesta apropiada.? Una de las formas es aprovechando las características de la aritmética entera. Recuerde que en esta aritmética no se generan decimales por lo tanto si nosotros tomamos un número y lo dividimos entre dos eso nos dará un resultado. Si ese resultado lo multiplicamos por dos ¿Nos volverá a dar el mismo número inicial...? Sí, pero solamente cuando este haya sido par ya que si hubiera sido impar al hacer la división entre dos se pierden sus decimales.

Vamos a hacerlo con un ejemplo: Si dividimos 7 entre 2 ¿cuánto nos da..? (Recuerda que son datos enteros y que estamos trabajando con aritmética entera por ser estos dos datos enteros) pues el resultado es 3 ya que en aritmética entera no se generan decimales. Y si ahora tomamos este resultado y lo multiplicamos por 2 nos va a dar 6 que no es igual al 7 inicial, por lo tanto podemos decir que como 6 no es igual a 7 entonces el 7 no es par.

Tal vez usted dirá Qué bobada...!! Si todos sabemos cuando un número es para o no. Pero no se olvide que el que va a ejecutar este algoritmo (convertido obviamente en programa es el computador y ese sí no fue a la escuela como nosotros).

Igualmente y para continuar con nuestro ejemplo si el número 8 es dividido entre 2 obtenemos el resultado 4 y si ese resultado es multiplicado por 2 obtenemos el 8 inicial. Por lo tanto podemos decir que 8 es un número par.

Luego para determinar si un número cualquiera es par todo lo que tenemos que hacer es dividirlo entre 2 y multiplicarlo por 2 si al final se obtiene el resultado inicial es porque el número es par. Si no se obtiene el resultado inicial es porque el número no es par.

De tal forma que nuestra pregunta

```

Sino
  Preguntar si el número es par

```

*Si lo es entonces escribir que el número leído es par*  
*Si no lo es escribir que el número leído no es par*  
 Fin

Se convierte en

Sino  
     *Si  $num / 2 * 2 = num$*   
         *Escriba "El número leído es par"*  
     Sino  
         *Escriba "El número leído no es par"*  
 Fin

Cuando se vaya a resolver la pregunta *Si  $num / 2 * 2 = num$*  no se olvide de la jerarquía de operadores para que el resultado sea el correcto y por ende la respuesta a dicha pregunta.

Entonces nuestro algoritmo que inicialmente era

*Algoritmo para determinar si un número es par*  
 Inicio  
     *Leer un número y guardarlo en una variable entera*  
     *Si ese número es negativo*  
         *Escribir que ese número no sirve para nuestro propósito*  
     Sino  
         *Preguntar si el número es par*  
         *Si lo es entonces escribir que el número leído es par*  
         *Si no lo es escribir que el número leído no es par*  
 Fin

Se ha convertido ahora en un algoritmo técnico así

*Algoritmo Número\_Par*  
 Variables  
     Entero : num  
 Inicio  
     Lea num  
     Si  $num < 0$   
         Escriba "El número debe ser positivo"  
     Sino  
         Si  $num / 2 * 2 = num$   
             Escriba "El número leído es par"  
         Sino  
             Escriba "El número leído no es par"  
 Fin

Cuál es la verdadera diferencia entre uno y otro..? Pues la diferencia es que la segunda versión de este algoritmo es absoluta y fácilmente codificable en un Lenguaje de Programación y la primera versión no es tan fácilmente codificable dado la gran cantidad de razonamientos que hay que