Programación

Unidad 1: Conceptos Básicos

"Si crees que puedes o crees que no puedes, tienes razón".

- Henry Ford

Introducción

¿Qué es programar?

Programar es planificar, es preparar de antemano las actividades que se van a desarrollar para llevar a cabo una tarea o resolución de problemas.

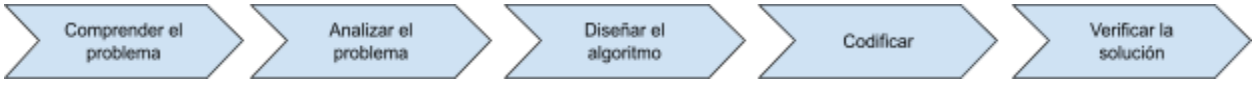
En esta materia aprenderemos a resolver problemas utilizando algoritmos, estructuras de datos y lenguajes de programación.

Ahora bien, ¿Qué es un problema? ¿Cómo llegamos a obtener la solución a dicho problema?

Definición	**¿Qué es un problema?** Podemos decir que un problema es el planteo de una situación a resolver mediante la aplicación de algún método.

El proceso de resolución de un problema con una computadora conduce al desarrollo de un programa y su ejecución en la misma. Si bien este proceso es esencialmente un proceso creativo, podemos considerar una serie de pasos comunes a seguir.

Pasos para la resolución de un problema



Comprender el problema

Este paso, aunque parezca una obviedad, no siempre se realiza de forma correcta y no se puede hallar la solución de algo que no comprendemos. En este paso debemos determinar cuál es el problema planteado. En nuestro caso la especificación o descripción del problema a resolver vendrá dada en forma de enunciado, por lo cual debemos leer la cantidad de veces que sea necesaria dicho enunciado hasta asegurarnos de haber comprendido “cuál es el problema”

Analizar el problema

Un problema consta de varios elementos que debemos identificar en este paso. Es por eso que para poder realizar un buen análisis del problema es conveniente poder responder a las siguientes preguntas:

1. ¿Cuál es el objetivo del problema?

La primer pregunta que nos deberíamos hacer para comenzar a resolver un problema es **"¿Qué quiero obtener?"** al resolver el problema. La respuesta a esta pregunta nos conduce a **identificar el objetivo** del problema. Nuevamente lo que se trata de definir en este punto es el **Qué**.

Tener en cuenta que el objetivo **no nos dice cómo** resolver el problema.
Ejemplo:

> Si mi problema consiste en "ir a Mar del Plata", entonces el objetivo (*el "Qué quiero obtener"*) es llegar a la ciudad de Mar del Plata. Ahora bien, si voy en auto, tren o micro eso sería "el cómo" y no formaría parte del objetivo.

2. ¿Qué datos de entrada se requieren?

Para responder a esta pregunta debemos identificar:

1. **Datos de entrada**: son los datos con los que se debe contar para poder resolver el problema. Por cada uno de estos datos debemos saber:
    a. **Su nombre**: a cada dato de entrada debemos definirle un nombre para poder identificarlo. Este nombre deberá ser representativo para que quede claro qué representa.
    b. **Su característica (Constante o Variable)**: esta clasificación dependerá de si el valor del dato de entrada puede cambiar o no durante la resolución del problema. Si el valor no cambia entonces lo clasificaremos como *Constante*, pero si el valor del dato de entrada puede variar durante la resolución del problema, lo clasificaremos como *Variable*.
    c. **Su tipo de dato**: el tipo de dato nos define cuál es el conjunto de valores posibles y qué tipo de operaciones puedo realizar sobre dicho dato de entrada. Los tipos de datos que manejaremos son:

Tipo de Dato	Valor posible
Entero (Integer)	Número Entero
Real (Float)	Número Real
Caracter (Character)	Un solo símbolo (letra, dígito, signo)
Cadena (String)	Conjunto de caracteres (palabra, frases)
Lógico (Boolean)	Verdadero o Falso

2. **Precondiciones**: las precondiciones indican qué valores podrán tomar los datos de entrada de tipo Variable para que se consideren válidos. Podríamos decir que las precondiciones imponen una restricción sobre los valores definidos por el tipo de dato.

3. ¿Cuál es la salida deseada?

En este punto debemos especificar cuál o cuáles son los resultados esperados y por cada uno definir el tipo de dato.

No es necesario definir nombre ni tampoco clasificarlo en constante o variable ya que siempre se los considera como variable.

4. ¿Lote de datos de prueba para verificar la solución?

En este punto nos centraremos en definir un conjunto de datos que nos permitan verificar la solución del problema. El lote de datos de prueba se compone de una lista de valores para cada uno de los datos de entrada. Podemos representarlo en forma de tabla de valores

Este lote de datos de prueba será utilizado más adelante, cuando hayamos diseñado la solución, para realizar una *Prueba de Escritorio*. Una prueba de escritorio es una herramienta que nos permitirá verificar si el programa produce la salida deseada.

Ejemplo:

"Desarrollar un procedimiento que permita ingresar el nombre de un alumno y las notas de 1er y 2do parcial y devuelva VERDADERO en caso de haber aprobado la materia o FALSO en caso de no aprobar"

Objetivo: saber si el alumno aprobó o no la materia
Datos de entrada:
nombre: Variable de tipo Cadena. Nombre del alumno.
notaParcial1: Variable de tipo Real. Nota del primer parcial
notaParcial2: Variable de tipo Real. Nota del segundo parcial
CANTIDAD_EXAMENES: Constante de tipo Entero cuyo valor es 2

Precondiciones:
notaParcial1 > 0 Y notaParcial1 <= 10
notaParcial2 > 0 Y notaParcial2 <= 10

Datos de salida:
alumno_aprobado: Lógico

Lote de Datos:

nombre	notaParcial1	notaParcial2	Salida
Leo	5	6	FALSO
Alejandro	8	6	VERDADERO
María	1	10	FALSO

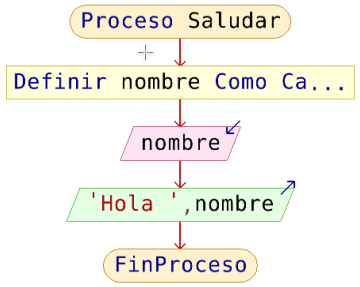
Diseñar el algoritmo

En el paso anterior, nos enfocamos en “Qué” tengo que hacer. En este paso, determinaremos **“Cómo”** resolver el problema. Es decir cómo vamos a procesar los datos de entrada para obtener la salida esperada.

Para diseñar nuestra solución podemos utilizar dos herramientas que son el *pseudocódigo* y el *diagrama de flujo*.

El diagrama de flujo (flowchart) es una representación gráfica de un algoritmo mientras que el pseudocódigo es una herramienta en la que las instrucciones se escriben utilizando palabras en español. Más adelante especificaremos cada una de ellas en detalle.

Ejemplos

Pseudocódigo	Diagrama de flujo
1 Proceso Saludar 2 Definir nombre como Cadena; 3 4 Leer nombre; 5 6 Mostrar "Hola ", nombre; 7 FinProceso	

En la próxima unidad veremos una técnica para poder diseñar un algoritmo.

Verificar la solución

La verificación de una solución es el proceso mediante el cual, con una amplia variedad de datos de entrada, llamados datos de prueba (test) se determina si la solución funciona correctamente o si tiene errores que deberán ser corregidos. El conjunto de datos de prueba debe contemplar

valores normales de entrada, valores extremos de entrada que permitan verificar los límites del programa y valores que permitan comprobar aspectos especiales del programa.

En esta instancia contamos con la *Prueba de Escritorio* como herramienta para verificar la solución. Para utilizar esta herramienta desarrollaremos una tabla con los valores que van tomando los datos de entrada y los datos de salida a por cada instrucción de nuestra solución.

Paso	Entrada_1	...	Entrada_N	Salida_1	...	Salida_N
1						
...						
N						

Codificar

La codificación consiste en traducir la solución diseñada en el paso anterior utilizando un *lenguaje de programación*. Dado que el diseño de una solución es independiente del *lenguaje de programación* a utilizar, debería poder ser escrito con igual facilidad en un lenguaje u otro.

Definición	**¿Qué es un lenguaje de programación?** Es un conjunto de instrucciones (acciones u operaciones) que nos permiten crear programas a ser ejecutados por una computadora.

Lenguajes de Programación

¿Qué es un lenguaje de programación?

Como mencionamos anteriormente, un lenguaje de programación es un conjunto de instrucciones (acciones u operaciones) que nos permiten crear programas a ser ejecutados por una computadora. Este conjunto de reglas nos permiten escribir instrucciones en un “idioma” que puede ser traducido a un lenguaje comprensible para la computadora.
Al lenguaje que comprende la computadora lo llamamos *lenguaje máquina*.

Como todo lenguaje, un lenguaje de programación, tiene su parte sintáctica y su parte semántica. Esto implica que todos los lenguajes de programación poseen reglas sobre cómo deben ser escritas las sentencias y de qué forma (*sintaxis*).

Todo lenguaje de programación posee un conjunto básico de instrucciones comunes que son:

- Instrucciones de entrada/salida. Permiten la transferencia de datos desde y hacia la computadora.
- Instrucciones de cálculo. Permiten realizar operaciones aritméticas.
- Instrucciones de control. Permiten modificar la secuencia de la ejecución del programa.

A su vez, los lenguajes de programación pueden clasificarse en 2 grandes grupos:

- lenguajes de bajo nivel
- lenguajes de alto nivel.

Lenguaje máquina

Es el lenguaje que entiende la máquina sin necesidad de traducciones. El problema es que desarrollar programas utilizando este lenguaje resulta una tarea sumamente difícil para una persona ya que sus instrucciones son secuencias de 0 (ceros) y 1 (unos), tales como 11110000, 011100110, etc, que son muy difíciles de recordar y manipular.

Ejemplo:

```
Mover el contenido del registro 4 al registro 8, se podía expresar como

4048 o bien 0010 0000 0010 1000
```

En consecuencia se necesitan lenguajes de programación “amigables” con las personas, que permitan escribir programas con mayor facilidad.

Lenguaje de bajo nivel

También llamados *ensambladores*, son aquellos lenguajes cuyas sentencias están formadas por códigos nemotécnicos (abreviaturas de palabras inglesas), por lo que sus instrucciones son mucho más comprensibles y fáciles de recordar que los lenguajes de máquina.

Algo a tener en cuenta es que estos lenguajes son dependientes de la arquitectura de cada procesador debido a que cada procesador ofrece su propio conjunto de instrucciones.

Ejemplo:

```
Mover el contenido del registro 4 al registro 8, se podía expresar como

MOV R5, R6
```

Un programa escrito utilizando algún lenguaje de bajo nivel, requiere de un ensamblador (assembler) que lo convierta a lenguaje máquina.

Lenguaje de alto nivel

Son aquellos cuyas instrucciones están formadas por palabras similares a las de los lenguajes naturales (comúnmente utilizado por humanos). Estos lenguajes son independientes del procesador y por lo tanto, resulta mucho más sencillo escribir un programa en un lenguaje de alto nivel para luego ser traducido a lenguaje máquina.

Algunos de los lenguajes de alto nivel más conocidos son C, C++, C#, Java, PHP, Python.

Más allá de que las instrucciones de estos lenguajes son fáciles de recordar y manipular, requieren ser traducidos a código máquina. Para realizar la traducción de lenguaje de alto nivel a lenguaje máquina se utilizan programas llamados *compiladores* y cada lenguaje posee el suyo, es decir que C tendrá su propio compilador, Java tendrá el suyo, etc.

Tipos de programación

No sólo existen distintos tipos de lenguajes de programación sino que también existen distintas formas de programar, cada uno con sus características. Los distintos tipos de programación son:

Secuencial

Este tipo de programación se basa en la construcción de programas cuyas sentencias se escriben y ejecutan de manera secuencial. Se utiliza la instrucción ***goto*** para realizar una bifurcación (división) en el flujo normal de ejecución del programa.

Estructurada

Este tipo de programación se basa principalmente en la modularidad de los programas. Esto quiere decir que cada programa se compone de módulos más pequeños donde cada uno realiza una tarea específica.

En otras palabras, lo que se trata de hacer es de dividir al problema en problemas más pequeños para poder alcanzar soluciones más simples.

Este tipo de programación define 3 reglas:

- El programa debe tener un diseño modular.
- Cada módulo es diseñado de modo descendente.
- Cada módulo es codificado utilizando las 3 estructuras de control básicas: secuencia, selección y repetición (no está permitido el uso de instrucciones ***goto***).

Lógica

La programación lógica es una forma de programar donde se deben definir hechos y relaciones entre los mismos, que conforman la llamada base de conocimiento del programa. Se utiliza el principio de razonamiento lógico para responder a las preguntas realizadas. Se basa en la lógica formal y en el cálculo de predicados de primer orden.

Orientada a objetos (POO)

Este enfoque se basa o guarda cierta analogía con la vida real. El desarrollo de software se basa en el diseño y construcción de objetos que interactúan entre sí para resolver el problema planteado. Cada objeto se compone de datos y acciones que manipulan esos datos.

Programación Estructurada

Como mencionamos anteriormente, la programación estructurada significa escribir un programa de acuerdo a las siguientes reglas:

- El programa debe tener un diseño modular.
- Cada módulo es diseñado de modo descendente.
- Cada módulo es codificado utilizando las 3 estructuras de control básicas: secuencia, selección y repetición (no está permitido el uso de instrucciones ***goto***).

La programación estructurada utiliza un número limitado de estructuras de control que minimizan la complejidad de los programas y por consiguiente ayuda a reducir los errores. Esto hace también que los programas sean más fáciles de escribir, leer y verificar, en otras palabras ayuda a la mejorar la mantenibilidad del código.

La programación estructurada incorpora lo que llamamos *estructuras básicas de control*, que nos permiten especificar y definir el orden en que se ejecutarán las instrucciones de un algoritmo. Estas estructuras son fundamentales en los lenguajes de programación y en el diseño de algoritmos. Las 3 estructuras básicas son:

- sentencia
- estructuras de control
- estructuras de repetición

Bibliografía utilizada

Luis Joyanes Aguilar, Ignacio Zahonero Martínez. Programación en C. Segunda Edición. España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA. S.A.U., 2005. ISBN-84: 481-9844-1.

● Brian W. Kernighan, Rob Pike. La práctica de la programación. Pearson Educación. Méjico (2000)

● Debugging with CodeBlocks: http://wiki.codeblocks.org/index.php?title=Debugging_with_Code::Blocks