1000 PROGRAMADORES

Introducción a la
Programación en Python

MÓDULO 1

Diseño de programas.
Fases en la resolución de problemas computacionales.
Algoritmo. Definición. Características.
Pseudocódigo.
Estructuras fundamentales.

Ministerio de Educación
Cultura, Ciencia y Tecnología
Gobierno de Salta

Ministerio de Economía
y Servicios Públicos
Gobierno de Salta

Universidad
Nacional de Salta

INTRODUCCIÓN

La informática es la ciencia que estudia el análisis y resolución de problemas utilizando computadoras .

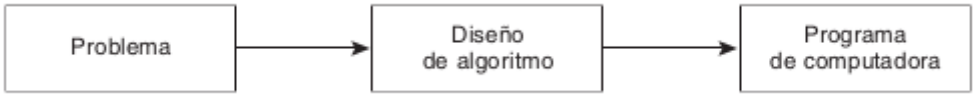
La palabra ciencia se relaciona con una metodología fundamentada y racional para el estudio y resolución de problemas. En este sentido la informática se vincula especialmente con la Matemática y la Ingeniería.

La resolución de problemas utilizando herramientas informáticas, llamados problemas computacionales, tiene aplicaciones en áreas muy diferentes como biología, comercio, control industrial, administración, robótica, educación, arquitectura, diseño, etc.

El mundo real es naturalmente complejo y en muchas ocasiones los problemas a resolver resultan difíciles de representar y modelizar para conseguir un programa que los resuelva.

El programador de computadora es antes que nada una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático.

En este curso nos referiremos a la metodología necesaria para resolver problemas mediante programas cuyo eje central es el concepto de algoritmo.
La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto.



DISEÑO DE PROGRAMAS

Definición: Un lenguaje de programación es la combinación de símbolos y reglas que permiten la elaboración de programas con los cuales la computadora puede realizar tareas o resolver problemas de manera eficiente.Los lenguajes de programación se clasifican en:

1. Lenguaje máquina. Las instrucciones son directamente entendibles por la computadora y no necesitan traductor para que la CPU (unidad de procesamiento central) pueda entender y ejecutar el programa. Utiliza un código binario (0 y 1), se basa en bits (abreviatura inglesa de dígitos binarios).
2. Lenguaje de bajo nivel (ensamblador). Las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos.
3. Lenguaje de alto nivel. Es semejante al lenguaje humano (en general en inglés), lo que facilita la elaboración y comprensión del programa. Por ejemplo Basic, Pascal, Cobol, Fortran, C, Python, Java, PHP, etcétera.

Definición: Se denomina algoritmo al conjunto de pasos ordenados y finitos que permiten resolver un problema o tarea específica. Los algoritmos son independientes del lenguaje de programación y de la computadora que se vaya a emplear para ejecutarlo. Todo algoritmo debe ser:

1. Finito en tamaño o número de instrucciones (tiene un primer paso y un último paso) y tiempo de ejecución (debe terminar en algún momento). Por lo tanto, debe tener un punto particular de inicio y fin.
2. Preciso. Debe tener un orden entre los pasos.
3. Definido. No debe ser ambiguo (dobles interpretaciones); si se ejecuta el mismo algoritmo el resultado siempre será el mismo, sin importar las entradas proporcionadas.
4. General. Debe tolerar cambios que se puedan presentar en la definición del problema. Toda actividad que realizamos la podemos expresar en forma de algoritmo.

Existen dos tipos de algoritmos, los que se desarrollan para ser ejecutados por una computadora, llamados algoritmos computacionales, y los que realiza el ser humano, es decir, algoritmos no computacionales; como ejemplos de éstos tenemos:

a) Cambiar un neumático de un automóvil.
b) Preparar unos panqueques.

Definición: Existen diferentes conceptos sobre lo que es un programa:

1. Es un algoritmo desarrollado en un determinado lenguaje de programación, para ser utilizado por la computadora; es decir, es una serie de pasos o instrucciones

ordenadas y finitas que pueden ser procesadas por una computadora, a fin de permitirnos resolver un problema o tarea específica.

2. Secuencia de instrucciones mediante las cuales se ejecutan diferentes acciones de acuerdo con los datos que se desee procesar en la computadora.

3. Expresión de un algoritmo en un lenguaje preciso que puede llegar a entender una computadora.

Como se comentó en la sección anterior, no todo algoritmo puede llegar a ser un programa de computadora, debido a que existen algunos algoritmos que requieren ser realizados físicamente. Los programas que puede ejecutar una computadora conllevan un proceso lógico.

Recordar: *Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa.*
Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así, por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizan sin importar el idioma del cocinero.

En la ciencia de la computación y en la programación, los algoritmos son más importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

ETAPAS EN EL DISEÑO DE UN PROGRAMA

Proponemos las siguientes etapas para diseñar un programa, aunque para algunos autores pueden describirse de maneras diferentes. Las etapas se describen a continuación.

1. Definición del problema
Esta fase la proporciona el enunciado del problema, el cual requiere una definición clara y precisa (no debe ser ambiguo). Es importante que se entienda perfectamente lo que pretendemos que haga la computadora para poder continuar con la siguiente etapa.

2. Análisis del problema
Una vez que se ha comprendido lo que se desea que la computadora haga, la etapa de análisis es muy importante ya que en ésta se identifican tres factores indispensables:

a. Qué información se necesita para obtener el resultado deseado (datos de entrada).

b. Qué información se desea producir (datos de salida).

c. Los métodos y fórmulas que se necesitan para procesar los datos y producir esa salida.

d. Diseñar caso/s de prueba: un caso de prueba está constituido por una colección de datos de entrada y las condiciones o restricciones que sobre ellos operan, necesarios para obtener la salida del diagrama que modela la solución del problema.

3. Diseño y técnicas para la formulación de un algoritmo
La etapa de diseño se centra en desarrollar el algoritmo basándonos en las especificaciones de la etapa del análisis; podemos representar un algoritmo mediante diagramas de flujo, diagramas de bloque o con pseudocódigo.
En este módulo usaremos pseudocódigo para definir nuestros algoritmos, en la siguiente sección detallaremos sus características.

4. Codificación
En la etapa de codificación se transcribe el algoritmo definido en la etapa de diseño en un código reconocido por la computadora; es decir, en un lenguaje de programación; a éste se le conoce como código fuente. Por ejemplo el lenguaje “Python” es un lenguaje de programación y es el que utilizaremos en el presente curso.

5. Prueba y depuración
La prueba consiste en capturar datos hasta que el programa funcione correctamente.

A la actividad de localizar errores se le llama depuración. Existen dos tipos de pruebas: de sintaxis y de lógica. Las pruebas de sintaxis se ejecutan primero, son las más sencillas y las realiza el compilador del programa cada vez que se ejecuta el programa hasta que el código no presente errores, es decir que la sintaxis que requiere el lenguaje sea la correcta, de lo contrario el propio compilador va mostrando los errores encontrados para que se modifiquen y se pueda ejecutar el código; estos errores pueden ser falta de paréntesis, o puntos y comas o palabras reservadas mal escritas.

Las pruebas de lógica son las más complicadas ya que éstas las realiza el programador; consisten en la captura de diferentes valores y revisar que el resultado sea el deseado, es decir el programador tendría que modificar el código hasta que el programa funcione correctamente.

6. Documentación

Es la guía o comunicación escrita que permite al programador o al usuario conocer la funcionalidad del programa.La documentación sirve para que el código fuente sea más comprensible para el programador o para otros programadores que tengan que utilizarlo, así como para facilitar futuras modificaciones (mantenimiento). Hay dos tipos de documentación:

• Interna. Se generan en el mismo código y generalmente es mediante comentarios.

• Externa. Son los manuales y es independiente al programa. También puede ser la ayuda en el mismo software.

7. Mantenimiento

Se dice que un programa no se termina al 100%, ya que es necesario hacer algún cambio, ajuste o complementación para que siga funcionando correctamente; para llevarlo a cabo se requiere que el programa esté bien documentado.

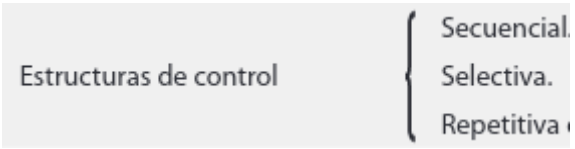
Muchos de los programas, aplicaciones y sistemas operativos tienen actualizaciones, por lo que surgen versiones diferentes. Por ejemplo: Windows 95-98-7, Ubuntu 12.04-16.04-18.04.

En este curso trabajaremos hasta la etapa 5 de Prueba y Depuración. Y en este módulo en particular haremos hincapié en las etapas 1, 2, 3 y una variante de la 7, realizando una prueba llamada "Prueba de Escritorio" para chequear el correcto funcionamiento de nuestros algoritmos.

PSEUDOCÓDIGO

Para diseñar los programas de computadora o en este caso los algoritmos, comúnmente se utilizan diferentes estructuras de control a fin de poder llegar a la solución de un problema, cuáles y cuántas dependen del problema mismo.

Estas estructuras controlan cómo se ejecutan los algoritmos, es decir el orden de las instrucciones, ya que tienen un punto de entrada y un punto de salida. Las podemos clasificar en:



Usaremos palabras específicas que denotan una instrucción en particular.

INGRESAR: para el ingreso/lectura de datos.

MOSTRAR: para la salida/impresión de datos.

ASIGNAR: para asociar un valor a una etiqueta.

SI-SINO: para plantear una o más alternativas, cada alternativa está formada por una condición lógica simple o compuesta. Una condición lógica puede tomar los valores Verdadero o Falso.

REPETIR…(cantidad)....VECES: para realizar una o varias acciones una cantidad de veces conocida.

REPETIR MIENTRAS (condición): para realizar una o varias acciones una cantidad de veces NO conocida. La repetición se realizará sólo si la condición impuesta es Verdadera y dejará de ejecutarse cuando dicha condición sea Falsa.

Ampliaremos estos conceptos a medida que vayamos avanzando y practicando el uso de pseudocódigo. Más abajo hay muchos ejemplos.

Estructura de control secuencial

Las instrucciones se ejecutan en orden, una por una desde la primera hasta la última, es decir el programa ejecuta todas las instrucciones del programa en el orden establecido sin saltarse ninguna de ellas.

Usaremos la instrucción de ASIGNACIÓN para ejemplificar una secuencia.

Problema 1: Calcular el área y el perímetro de un rectángulo, para lo cual se deben ingresar el valor del lado A y el valor del lado B, ambos números reales.

1. Definición del problema

- ➔ ¿El enunciado es preciso?
- ➔ ¿Es posible calcular el área y perímetro de un rectángulo?

2. Análisis del problema

- ➔ ¿Cuáles son los datos de entrada y salida?
- ➔ ¿Son suficientes los datos de entrada para resolver el problema?
- ➔ ¿Conocemos las fórmulas para obtener el área y el perímetro?
- ➔ ¿Somos capaces de ejemplificar con algunos valores de entrada cuáles serían los valores de salida?
- ➔ ¿Qué tipo de instrucciones necesitamos para diseñar el algoritmo?

Datos de Entrada: Dos números reales llamados LADO_A y LADO_B.
Datos de Salida: Valor de AREA y PERIMETRO, que serán reales.

Caso de Prueba 1:
Datos de Entrada: LADO_A = 4; LADO_B = 8
Datos de Salida: AREA = 32
Datos de Salida: PERIMETRO = 24

Caso de prueba 2:
Datos de Entrada: LADO_A = 3; LADO_B = 7
Datos de Salida: AREA = 21
Datos de Salida: PERIMETRO = 20

Las instrucciones que debemos usar se relacionan con:

- *Instrucciones para el ingreso de datos*
- *Instrucciones para la asignación de datos*
- *Instrucciones para la salida de datos*

3. Diseñar el algoritmo

Paso 1. Ingresar un valor para LADO_A
Paso 2. Ingresar un valor para LADO_B
Paso 3. Asignar a AREA el resultado de: LADO_A * LADO_B.
Paso 4. Asignar a PERIMETRO el resultado de: 2*LADO_A + 2*LADO_B.
Paso 5. Mostrar el valor de AREA.
Paso 6. Mostrar el valor de PERIMETRO.
Fin.

4. Prueba de Escritorio

LADO_A	LADO_B	ÁREA	PERÍMETRO	SALIDA
4	8	32	24	**32** **24**
3	7	21	20	**21** **20**
8,5	2	17	21	**17** **21**

Estructura de control selectiva o alternativa

De acuerdo con una condición que puede ser verdadera o falsa se elige una opción, la cual realiza una acción (una o varias instrucciones). La condición puede ser simple o compuesta (una o varias).

Problema 2: Dado un número real indicar si es positivo o no.

Datos de Entrada: Un número real llamado X
Datos de Salida: Un mensaje que indique “es positivo” o indique “no es positivo”

Caso de Prueba 1:
Datos de Entrada: X = 4;
Datos de Salida: “es positivo”

Caso de prueba 2:
Datos de Entrada: X = 0;
Datos de Salida: “no es positivo”

Las instrucciones que debemos usar se relacionan con:

- *Instrucciones para el ingreso de datos*
- *Instrucciones de alternativa*
- *Instrucciones para la salida de datos*

3. Diseñar el algoritmo

```
Paso 1.     Ingresar un valor para X
Paso 2.     si (X>0) entonces
Paso 2.a.           Mostrar "es positivo"
            sino
Paso 2.a.           Mostrar "no es positivo"
            Fin.
```

NOTA: El paso 3.a. está "repetido" a propósito, dado que se ejecutará un único paso 3.a. ya sea cuando la condición es verdadera o cuando la condición es falsa. NUNCA se ejecutarán ambos a la vez.

4. Prueba de Escritorio

X	SALIDA
4	**"es positivo"**
0	**"no es positivo"**

Problema 3: Dado un número real indicar si es positivo, negativo o nulo.

Datos de Entrada: Un número real llamado X
Datos de Salida: Un mensaje que indique "es positivo" o "no negativo" o "es nulo"

Caso de Prueba 1:
Datos de Entrada: X = 4;
Datos de Salida: "es positivo"

Caso de prueba 2:
Datos de Entrada: X = 0;
Datos de Salida: "es nulo"

Las instrucciones que debemos usar se relacionan con:

- *Instrucciones para el ingreso de datos*
- *Instrucciones de alternativa*
- *Instrucciones para la salida de datos*

3. Diseñar el algoritmo

```
Paso 1.     Ingresar un valor para X
Paso 2.     si (X>0) entonces
Paso 2.a.           Mostrar "es positivo"
            sino si (X<0) entonces
Paso 2.a.           Mostrar "es negativo"
            sino
Paso 2.a.           Mostrar "es nulo"
            Fin.
```

NOTA: ¿Es posible usar otra combinación de alternativas y encontrar el mismo resultado?

4. Prueba de Escritorio

X	SALIDA
4	**"es positivo"**
0	**"es nulo"**

Estructura de control repetitiva

Una acción se repite una cantidad definida o indefinida de veces mientras una condición sea verdadera.

Problema 4: Dados N números reales indicar cuántos de ellos son positivos.

Datos de Entrada: Cantidad de números reales (N) y Lista de números reales (X)
Datos de Salida: Cantidad de reales positivos de la lista (Cant)

Caso de Prueba 1:
Datos de entrada:
N= 5
X = 10, 50, -3, 0, 8
Dato de salida:
Cant = 3

Caso de prueba 2:

Datos de entrada:

N= 6

X = 0, -50, -3, 0, -8, -1

Dato de salida:

Cant = 0

Las instrucciones que debemos usar se relacionan con:

- *Instrucciones para el ingreso de datos*
- *Instrucciones de alternativa*
- *Instrucciones para la repetición de tareas*
- *Instrucciones para la salida de datos*

3. Diseñar el algoritmo

Paso 1. Asignar a Cant el valor 0
Paso 2. Ingresar un valor para N
Paso 3. Repetir N veces
Paso 3.a. Ingresar un valor para X
Paso 3.b. si (X>0) entonces
Paso 3.b.1. Asignar a Cant el resultado de sumar Cant con 1
Paso 4. Mostrar Cant
Fin.

4. Prueba de Escritorio

Cant	N	X	SALIDA
0	5		
1		10	
2		50	
		-3	
		0	
3		8	**3**

Problema 5: Dada una cantidad desconocida de números reales no nulos, indicar cuántos de ellos son positivos y cuántos números fueron ingresados en total.

Datos de Entrada: Lista de números reales (X)
Datos de Salida: Cantidad de reales positivos de la lista (Cant), Cantidad de reales de la lista (CantTot)

Caso de Prueba 1:
Datos de entrada:
X = 10, 50, -3, -1, 8, 0

Dato de salida:
Cant = 3
CantTot = 5

Caso de prueba 2:
Datos de entrada:
X = -50, -3, -1, -8, 0

Dato de salida:
Cant = 0
CantTot = 4

Las instrucciones que debemos usar se relacionan con:

- *Instrucciones para el ingreso de datos*
- *Instrucciones de alternativa*
- *Instrucciones para la repetición de tareas*
- *Instrucciones para la salida de datos*

3. Diseñar el algoritmo

Paso 1. Asignar a CantTot el valor 0
Paso 2. Asignar a Cant el valor 0
Paso 3. Ingresar un valor para X
Paso 4. Repetir mientras X no sea nulo
Paso 4.a. Asignar a CantTot el resultado de acumular CantTot con 1
Paso 4.b. si (X>0) entonces
Paso 4.b.1. Asignar a Cant el resultado de sumar Cant con 1
Paso 4.c. Ingresar un valor para X
Paso 5. Mostrar Cant
Paso 6. Mostrar CantTot

Fin.

4. Prueba de Escritorio

CantTot	Cant	X	SALIDA
0	0		
1	1	10	
2	2	50	
3		-3	
4		-1	
5	3	8	
		0 *<Fin de entrada>*	**5** **3**

En los módulos siguientes profundizaremos sobre la definición formal de una estructura de control, su clasificación, similitudes, diferencias y muchos más ejemplos.

Ten siempre presente que la programación de una solución requiere de lápiz y papel antes que una computadora y un lenguaje de programación. Hay que idear la estrategia antes de disputar el partido.

BIBLIOGRAFÍA

[1] Diseño de Algoritmos. M.A. Corona Nakamura - M.A. Ancona Valdez. Mc Graw Hill. 2011
[2] Fundamentos de Programación 4ta Edicion. L. Joyanes Aguilar. Mc Graw Hill. 2008
[3] Metodología de la Programación 3ra Edición. O. Cairó. Alfaomega. 2005