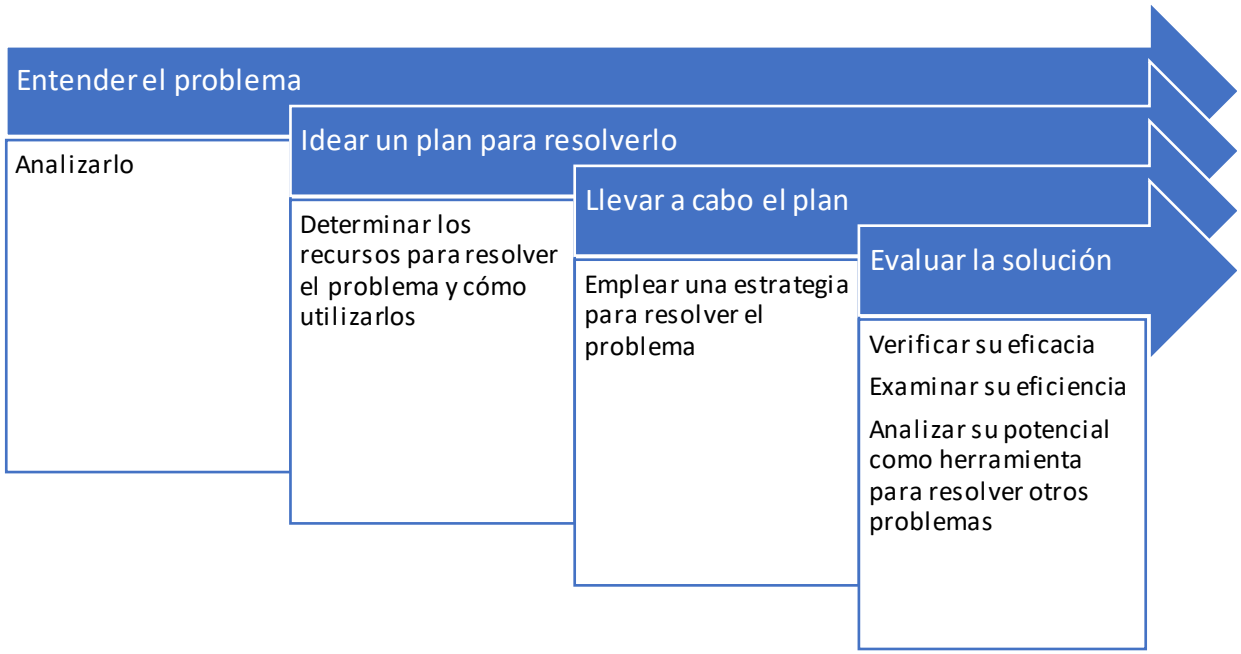
Algoritmia y Programación

Principios, convención y recomendaciones

Programación de computadoras

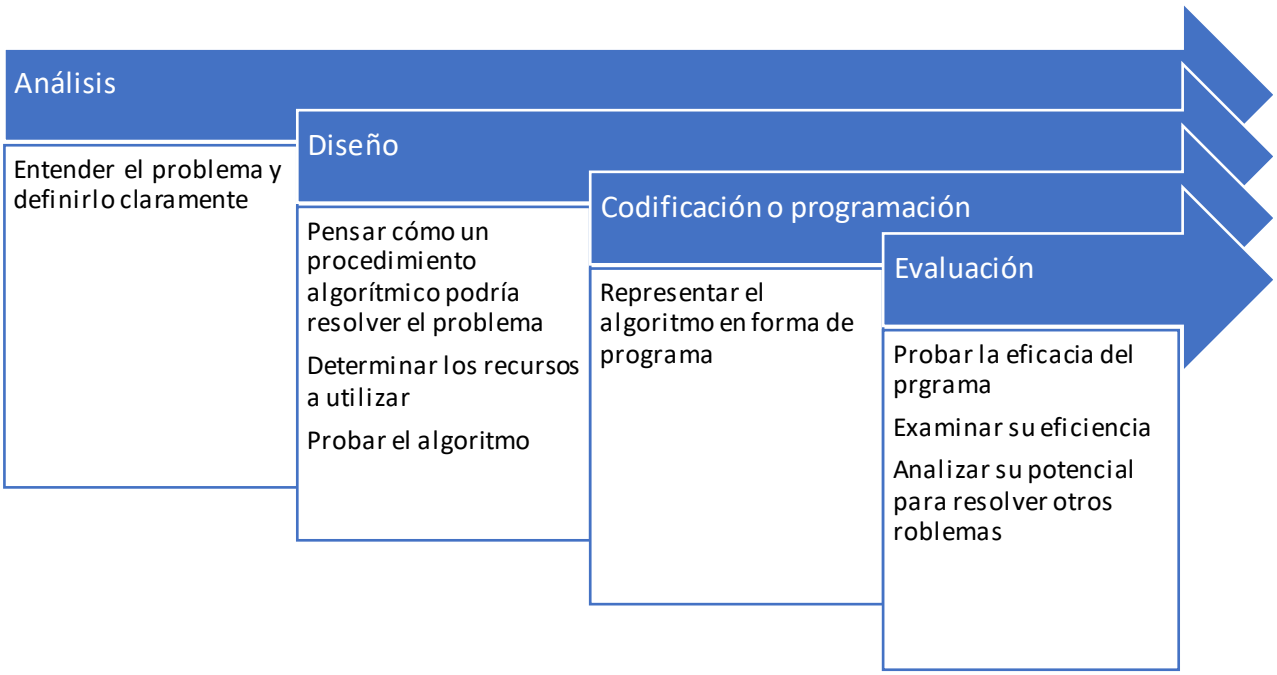
Como un programa es un *algoritmo* representado en un *lenguaje de programación* para que pueda ejecutarlo una *computadora*, desarrollar un programa implica descubrir la solución al problema que debe resolver el programa, es decir, descubrir el algoritmo, y luego representarlo en forma de programa. La actividad más importante y difícil en el proceso de creación de software es descubrir los algoritmos, por lo que la algoritmia debe basarse necesariamente en el proceso de resolución de problemas, un proceso imaginativo y artístico cuyas fases, definidas a grandes rasgos por el matemático G. Pólya a fines de los años cuarenta siguen siendo los principios en que se basan los métodos actuales para resolver problemas.

Las fases de Pólya para resolver problemas son:



Estas fases no tienen por qué seguirse linealmente, sino más bien deben completarse en algún momento durante el proceso de resolución, que en general implica repasar una y otra vez las fases en forma evolutiva, retrocediendo algún paso de ser necesario, hasta que la solución sea satisfactoria.

Para la creación de programas, las fases anteriores pueden traducirse a:



La estrategia más común para desarrollar algoritmos es la denominada "divide y vencerás", que consiste en descomponer el problema principal en una lista de problemas más simples, que a su vez se siguen descomponiendo hasta llegar a un nivel en que los problemas sean resolubles por acciones simples de la computadora. Este proceso se denomina de **refinamientos sucesivos**, y es similar al que se emplea para escribir artículos o libros partiendo de borradores; para la programación, también se lo llama de diseño descendente, porque parte de ideas generales y desciende hasta los detalles.

El resultado de los refinamientos sucesivos es un conjunto de instrucciones paso a paso que, al completarse, resuelve el problema original, o sea, es un **algoritmo**, expresado en términos de acciones esenciales llamadas primitivas de programación no ejecutables, en un lenguaje híbrido entre un lenguaje de programación y un lenguaje natural llamado pseudocódigo, o código de documentación interna de programas.

Los enunciados de problemas a resolver, en cualquier nivel de refinamiento, deben observar la definición de algoritmo: deben ser ejecutables por una computadora, y ser no ambiguos. Respecto a las capacidades de las computadoras, se analizan a continuación, y en cuanto a la no ambigüedad, para que se cumpla este precepto es imprescindible que los enunciados sean libres de contexto, es decir, que sean autocontenidos (independientes de cualquier otro enunciado) y que se interpreten de una única manera posible.

Capacidades de una computadora

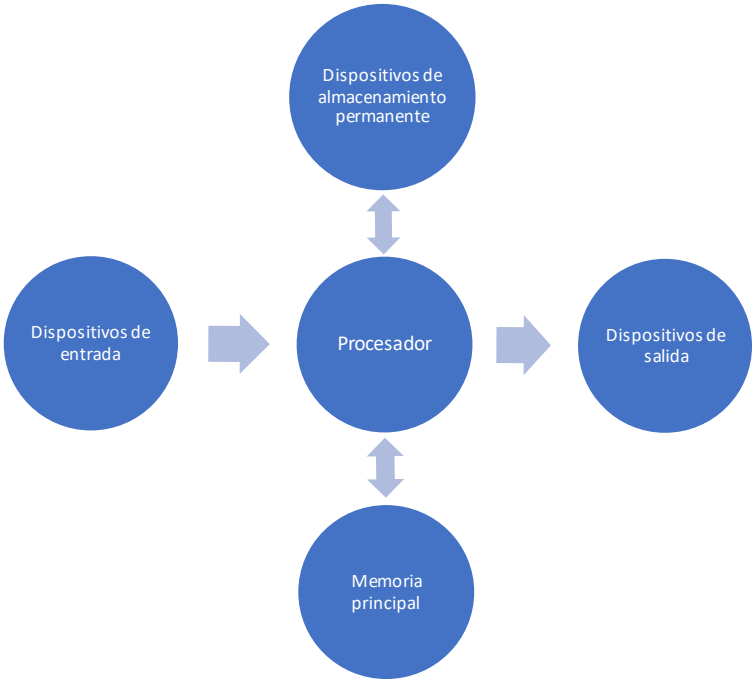
Para que una máquina pueda ejecutar algoritmos basta que pueda:

- Recibir entradas (aceptar información que le proporcione el usuario)
- Almacenar y recuperar información (mover y almacenar información en su memoria interna o en dispositivos de almacenamiento masivo)
- Procesar información (llevar a cabo operaciones aritméticas o lógicas -toma de decisiones- con la información)
- Producir salidas (proporcionar información al usuario)

En el procesamiento de datos con cualquier computadora se realizan los siguientes procesos:

Por lo tanto, para que los enunciados de la solución de un problema se puedan convertir en un programa, sólo pueden referirse a alguna de estas actividades. Cada una de estas actividades se asocia a dispositivos de hardware específicos de la computadora.

Componentes de Hardware asociados a las Capacidades de la Computadora



Los dispositivos de entrada aceptan entradas del exterior de la computadora. El dispositivo más común es el teclado QWERTY (llamado así por la primera fila de teclas de letras), aunque las computadoras pueden aceptar señales de entrada de muchos otros dispositivos como los dispositivos apuntadores: ratones (*mice*), bolas rastreadoras (*trackballs*), palancas de mando (*joysticks*), o dispositivos de lectura: lectoras ópticas de códigos de barra o de caracteres, para leer códigos universales de productos (UPC: *universal product codes*) o efectuar reconocimiento óptico de caracteres (OCR: *optical character recognition*) respectivamente.

La memoria central y los dispositivos de almacenamiento sirven para almacenar la información y los programas o conjunto de órdenes para que la computadora mueva, almacene, produzca, acepte o proporcione información. Se emplean diferentes tipos de memoria para tareas de almacenamiento a corto o largo plazo; así, en la memoria central se almacenan o "cargan" los programas que la computadora deba ejecutar de inmediato y la información necesaria para ejecutar cada orden o instrucción del programa en su oportunidad; y en la memoria auxiliar o masiva, representada por los dispositivos de almacenamiento (los dispositivos de almacenamiento más comunes son los discos rígidos, los disquetes, los discos compactos y las cintas), se almacenan programas e información en forma permanente para su libre disponibilidad en cualquier momento. Metafóricamente, se podría decir que la memoria central es el "escritorio de trabajo" de la computadora, y que los dispositivos de almacenamiento son sus "bibliotecas" o repositorios de programas e información.

Un procesador o unidad central de procesamiento (UCP) (en inglés CPU), es el dispositivo que procesa información, llevando a cabo todos los cálculos aritméticos y tomando decisiones básicas con base en los valores de la información y conforme a las órdenes de un programa. La UCP es, de hecho, el "cerebro" de la computadora.

Los dispositivos de salida envían información al exterior de la computadora. Los dispositivos más comunes son monitores e impresoras. Los monitores, también llamados monitores de vídeo o pantallas de vídeo *(VDT: video display terminal)* no sólo sirven para que la computadora exhiba información al mundo exterior sino también para que el usuario vea los caracteres que está ingresando a la computadora al momento de teclearlos; en ambos casos, se dice que la información presentada es de "copia efímera" *(soft copy)*, puesto que sólo puede visualizarse mientras se encuentre representada en la pantalla, y se los clasifica como dispositivos de copia efímera. Los monitores, en la actualidad suelen ser de pantalla plana de cristal líquido *(LCD: liquid crystal display)* o de diodos emisores de luces *(LED: light-emitting diode)*. Las impresoras permiten obtener copias físicas en papel de cualquier información, por lo que la información que producen es de "copia permanente" (*hard copy*); así es que desde el punto de vista de la información que presentan, las impresoras se clasifican como dispositivos de "copia permanente".

Almacenamiento de datos en la memoria central en Python

Los datos en cualquier lenguaje de programación se representan y referencian mediante recursos denominados variables. En Python se pueden usar variables cuyos nombres o identificadores deben comenzar con una letra y pueden comprender letras (en minúscula o mayúscula, incluso ñ o Ñ, y vocales acentuadas), dígitos decimales y el carácter de subrayado. El tamaño de los nombres es ilimitado, y el lenguaje es sensible a mayúsculas y minúsculas.

En Python los datos que se obtienen del teclado, que es el dispositivo de entrada estándar, se almacenan como cadenas de caracteres. Para obtener un dato desde el teclado se debe usar la función estándar *input()* y asignarle lo que devuelva esa función a una variable:

```
dato_cadena=input('Ingrese un dato: ')
```

donde el texto entre paréntesis y encerrado entre comillas simples (también se puede usar indistintamente comillas dobles) es lo que se mostrará en el dispositivo de salida estándar, es decir, la pantalla, para pedir el dato.

Si se quiere trabajar con números, se debe pedir explícitamente que se convierta el valor de una variable que contiene una cadena de caracteres a uno de los tipos numéricos del lenguaje. Los tipos de números que se pueden manipular en Python son enteros *(int)*, reales *(float)* o complejos *(complex)*. Por ejemplo, si se quiere convertir dato_cadena a un número entero, se debe usar la función de conversión *int()*:

```
dato_entero=int(dato_cadena)
```

También se puede abreviar todo el proceso, incluso ahorrándose el uso de una variable, componiendo funciones:

```
dato_entero=int(input('Ingrese un número entero: ')
```

Luego se puede usar la variable dato_entero en expresiones aritméticas.

Las funciones equivalentes para convertir valores a datos reales o complejos son *float()* y *complex()*, respectivamente.

La función para mostrar salidas en pantalla es *print()*, y admite como parámetros a variables de cualquier tipo, ya que hace la conversión automática a cadenas de caracteres.

Para los números reales, la parte entera se separa de la fraccionaria con punto en lugar de coma. Por ejemplo, para asignar un valor real a una variable, se puede escribir:

```
dato_real=1.5
```

Y los números complejos se deben especificar como una suma de la parte real más la imaginaria, que pueden ser enteras o reales. Por ejemplo, un complejo -1+2i debe ingresarse por teclado como -1+2j (usando j en vez de i, y sin usar espacios alrededor del operador central + o -), y si se pide imprimir una variable con este valor, se representará en pantalla encerrando la suma aritmética de partes entre paréntesis: (-1+2j).

Modelo de un programa tipo

En general, la solución de cualquier problema por parte de una computadora responde a un esquema tipo.

Como la planificación de la solución implica la determinación de recursos a utilizar, en general, un programa consta de una sección declarativa en la que se designan y definen los recursos que se emplean en el programa. Todos los valores que se van a procesar en el programa deben tener un nombre y un dominio, como las variables de una función matemática, de ahí que los recursos fundamentales de los programas se denominen precisamente *variables*. También se puede hacer referencia a valores determinados que se usen recurrentemente en un programa mediante un nombre, como por ejemplo pi o e, para lo cual se deben definir como recursos denominados *constantes*, a los que también se les asigna un nombre, y como dominio un valor constante.

Luego de determinados los recursos principales a emplear en la solución de un problema en forma de programa, sigue el desarrollo de la solución mediante *refinamientos sucesivos*. Por tanto, todo programa consta de una segunda sección llamada sección algorítmica o de código, en la que se procede a descomponer el problema a resolver en subproblemas de menor complejidad, representados por *enunciados*. Esta descomposición se efectúa en sucesivos pasos que determinan enunciados jerárquicamente dependientes de los anteriores, hasta llegar a enunciados de acciones esenciales primitivas. La jerarquización de los enunciados se denota mediante el sangrado o un esquema de itemización; de este modo, un programa se puede visualizar como un texto con enunciados con distintas sangrías según su nivel de refinamiento o con distintos niveles de itemización según su nivel de refinamiento.

Como todo programa debe quedar representado en un texto, el desarrollo de programas se hace trabajando en un editor de textos; de manera que refinar la solución de un problema implica simplemente abrir líneas debajo de su enunciado, y con un nivel de sangrado mayor a su enunciación o abrir una estructura de ítems de subproblemas según su jerarquía, es decir, descomponer el problema en una lista de enunciados de problemas más simples. Cuando todo problema a resolver por un programa haya quedado reducido a enunciados de acciones primitivas (que pueden expresarse en un lenguaje de programación), para concluir con la sección algorítmica o de código, y habilitar la sección evaluativa o de prueba se debe efectuar una prueba del algoritmo: se asignan distintos valores testigo a los datos que debe procesar, y se verifica que los resultados a obtener sean correctos.

La formalización de un programa o programación del algoritmo que soluciona un problema implica un último refinamiento expresando las acciones primitivas no ejecutables con la sintaxis del lenguaje de programación para obtener las primitivas de programación ejecutables. Entonces, los enunciados que realmente ejecutará la computadora serán los de la última jerarquía de cada problema.

Una primera descomposición de cualquier problema a resolver con un programa puede asimismo considerarse como un modelo o tipo. En general, en todo programa se debe efectuar una presentación de su objetivo y, eventualmente, condiciones o limitaciones para su ejecución al usuario que lo ejecute, y preparar los recursos o datos para resolver el problema. Esta primera parte de todo programa se puede denominar prólogo. Luego se debe proceder a la resolución del problema en sí, en una segunda parte, el verdadero cuerpo del programa, que se puede denominar resolución. Finalmente, en la mayoría de los programas se debe exhibir resultados, por lo que en general aparecerá una tercera parte o epílogo.

Modelo de un Programa Tipo en lenguaje Python

```
'''
SECCIÓN DECLARATIVA (descripción del problema, análisis basado en casos de prueba, y definición de recursos para resolverlo - documentación general del programa)
'''
# SECCIÓN ALGORÍTMICA (desarrollo y representación como programa de la solución)
# 1 Prólogo
...
# 2 Resolución (descomposición en subproblemas)
...
# 3 Epílogo
...
```

En Python, los enunciados de documentación de más de una línea se delimitan con tres comillas verticales simples (''') o dobles ("""), y los que ocupan completamente o finalizan una línea deben comenzar con el símbolo de número (#).

Cuando se escribe un programa en lenguaje Python o en cualquier otro lenguaje de programación, nunca se piensa para qué computadora se está escribiendo (procesador o tamaño de celdas de memoria) ni en qué sistema operativo tiene, porque el programa podría ejecutarse en cualquier computadora que tenga el ambiente integrado de desarrollo del lenguaje *(**I**ntegrated **D**evelopment **E**nvironment IDE)* o el ambiente integrado de desarrollo y aprendizaje Python *(**I**ntegrated **D**evelopment and **L**eaming **E**nvironment IDLE)*. Los programas escritos en lenguajes de programación de alto nivel tienen la apreciable característica de ser portables de una máquina a otra, porque los ambientes integrados de desarrollo de los lenguajes tienen un traductor de programas, que traduce programas escritos en el lenguaje (p.e. archivos con extensión PAS en el caso de Pascal, y con extensión PY en el caso de Python) a programas con órdenes propias de la computadora en la que está el ambiente. Por eso, las empresas que proveen los ambientes de desarrollo (como Borland, que desarrolló el IDE Turbo Pascal o la *Python Software Foundation*[1] que administra la licencia de código abierto de Python aprobada por OSI *(**O**pen **S**ource **I**nitiative)*, por lo cual es libremente utilizable y distribuible, incluso para uso comercial), tienen versiones de los mismos ambientes para distintas máquinas, de manera que cualquier programa funcione de igual forma en cualquier máquina.

Recomendaciones de estilo y de procedimiento para el desarrollo de programas

Los programas se pueden codificar siguiendo algún estilo; algunos estilos son más legibles que otros. Siempre es una buena idea facilitar que otros puedan entender fácilmente un programa, y adoptar un buen estilo de codificación contribuye enormemente a esto.

A continuación se enuncian algunas recomendaciones:

- Use sangría de 4 espacios.

[1] *Python Software Foundation* es una organización dedicada al avance de la tecnología de código abierto relacionada con el lenguaje de programación Python. La misión de *Python Software Foundation* es promover, proteger y avanzar en el lenguaje de programación Python, y apoyar y facilitar el crecimiento de una comunidad diversa e internacional de programadores de Python.

4 espacios son un buen compromiso entre una pequeña sangría (permite una mayor profundidad de anidación) y una gran sangría (más fácil de leer).

- Ajuste las líneas para que no superen los 79 caracteres.

  Esto ayuda a los usuarios con pantallas pequeñas y permite tener varios archivos de código uno al lado del otro en pantallas más grandes. Cuando por el nivel de sangría una línea superara este tamaño, se puede partir usando el carácter **** después de una coma o antes de un operador aritmético.

- Use líneas en blanco para separar bloques de código que representen la solución de un problema específico, usando un comentario de línea como encabezado para enunciar el problema que se resuelve a continuación.
- Cuando sea posible, coloque los comentarios en líneas exclusivas, es decir, evite terminar líneas de código con comentarios finales descriptivos a menos que resulte imprescindible.
- Use nombres identificadores de variables que representen el propósito de éstas para facilitar la comprensión del código y evitar confusiones.
- Evite reutilizar variables para distintos propósitos.
- Evite componer o anidar funciones para que el código no le quede críptico.
- Use la plantilla modelo que se proporciona para crear programas y complétela en forma consistente. Baje a su computadora o teléfono (el o no es exclusivo, es decir, a todo dispositivo en el que vaya a trabajar con Python) el archivo **programa.py** y ábralo desde el IDLE para guardarlo inmediatamente con el nombre del programa que vaya a desarrollar. Luego continúe con la edición para desarrollar el programa.