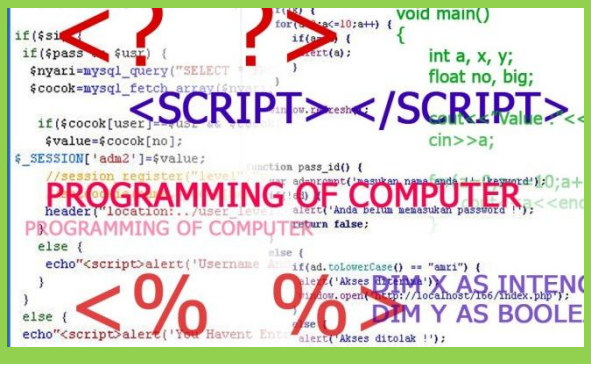
# INICIACIÓN A LA PROGRAMACIÓN II



# 1. Caracteres

Un carácter es cualquier símbolo en un ordenador. Pueden ser números, letras, puntuaciones, espacios, etc. Un carácter corresponde, por lo general, a un byte, conformado por ocho bits.

Los ordenadores pueden representar a un número finito de caracteres, los cuales se corresponden con los símbolos más usados para escribir por los seres humanos. Se clasifican en:

Letras minúsculas del alfabeto: { a, b, c, ..., x, y, z }

Letras mayúsculas del alfabeto: { A, B, C, ..., X, Y, Z }

Números (dígitos del Sistema Decimal): { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Caracteres especiales: { +, -, *, /, @, #, ñ, Ñ, á, é, ... }

Caracteres gráficos: { ♣, ♦, ♥, ♠, ... }

Caracteres de control: { Salto de línea, Tabulador horizontal, avance de página, ... }

- Caracteres especiales

Cada lenguaje de programación de computadoras tiene una combinación de caracteres especiales y habituales, normalmente alfanuméricos. Aunque el significado de cada uno cambia un poco entre lenguajes, el significado de los caracteres especiales sí suele ser específico del lenguaje. Además de las diferencias de significado, los caracteres especiales suelen requerir un tratamiento especial dentro de las sentencias de codificación de computadora. El uso incorrecto o inadecuado puede hacer que un programa dé resultados erróneos, que no funcione correctamente o que no funcione en absoluto.

Los caracteres especiales y su uso varían según el lenguaje de programación. Un carácter especial son las comillas, que según el lenguaje de programación pueden llegar a diferenciarse en simple **(')**, la doble **(")** y la invertida **(`)**. Cada una tiene un fin determinado y es necesario saber utilizarlas para no desesperar a la hora de la programación y evitar repetitivos errores.

Otros caracteres especiales son, por ejemplo, **+, *, ?** y **{ }.**

Así, el caracter **+** suele indicar que lo que tenemos a la izquierda, puede encontrarse una o más veces. El caracter ***** es similar a +, pero en este caso lo que se sitúa a su izquierda puede encontrarse cero o más veces.

El caracter **?** suele indicar opcionalidad, es decir, lo que tenemos a la izquierda puede o no aparecer (puede aparecer 0 o 1 veces).

Finalmente las llaves **{ }** sirven para indicar el número de veces exacto que puede aparecer el carácter de la izquierda, o bien un rango de veces que puede aparecer. Por ejemplo {3} indicaría que tiene que aparecer exactamente 3 veces, {3,8} indicaría que tiene que aparecer de 3 a 8 veces.

Los caracteres especiales pueden ser un único carácter, o pueden consistir en dos o más caracteres juntos. Por ejemplo, en C, en Java, en Python, son caracteres especiales las combinaciones formadas por una barra inversa seguida de una letra o una combinación formada por varios dígitos (denominadas también secuencias de escape), como las que aparecen a continuación:

| Código | Significado |
|---|---|
| `'\n'` | nueva línea (ir al principio de la siguiente línea) |
| `'\t'` | tabulador horizontal |
| `'\b'` | retroceso |

Los caracteres especiales son de gran importancia ya que su uso incorrecto pueden dar problemas de discordancias de datos y errores de sintaxis, lo que provocará que el programa no interprete bien las sentencias y no funcione.

## 2. Identificadores

Un identificador es simplemente el nombre que un programador da a una variable, constante o función.

Los identificadores pueden ser combinaciones de letras y números. Cada lenguaje tiene sus propias reglas que definen como pueden estar construidos. Cuando un identificador se asocia a una entidad concreta, entonces es el "nombre" de dicha entidad, y en adelante la representa en el programa.

Existen ciertas reglas a la hora de nombrar estos identificadores, según el tipo de lenguaje. En la mayoría de los lenguajes, los identificadores pueden comenzar con una letra o un signo de subrayado, y a continuación contener una combinación de letras, signo subrayado y números. No pueden contener espacios u otros caracteres, y en algunos lenguajes no pueden exceder de más de 8 caracteres. En algunos lenguajes de programación es usual que los identificadores compuestos de varias palabras se formen uniendo estas palabras con guión bajo y que tengan las letras iniciales en mayúscula.

Por ejemplo en lenguaje C:

Para que un identificador sea válido debe:
- Iniciar con una letra del alfabeto inglés, o con el signo (_)
- No debe contener caracteres especiales, tales como @, $, #
- Después de la primera letra puede contener más letras del alfabeto inglés, números, o el carácter (_)
- NO DEBE haber espacios en blanco en los identificadores
- C diferencia mayúsculas de minúsculas, entonces no es lo mismo declarar la variable *numero* que *Numero* **o** *NuMeRo*
- Existen palabras propias del lenguaje (palabras reservadas) que no pueden ser usadas como identificadores ej: if, do

- **EJEMPLOS:** Identificadores **válidos** definidos por el programador son:

```
numero
dia_del_mes
PINGUINO1
_ciudad
Z
```

- **EJEMPLOS:** Identificadores **no válidos** son:

```
123
_DÍA
numero*
lugar de nacimiento
año
```

## 3. Palabras clave o reservadas

En los lenguajes de programación existen una serie de indicadores reservados, con un significado especial y una finalidad determinada, que no se pueden usar con otro propósito, y que no podemos pues utilizar como identificadores para nombrar variables, etc

Cada lenguaje tiene un determinado número de palabras reservadas dependiendo de su complejidad. Algunos ejemplos de palabras reservadas son **If** , **for**, **while**, etc…

- Son palabras reservadas en Python:

| | | |
|---|---|---|
| and | except | is |
| elif | import | return |
| global | print | def |
| or | class | for |
| assert | exec | while |
| else | in | try |
| if | raise | del |
| pass | continue | from |
| break | finally | not |

- . Son palabras reservadas en Java:

| | | | |
|---|---|---|---|
| abstract | double | int | strictfp** |
| boolean | else | interface | super |
| break | extends | long | switch |
| byte | final | native | synchronized |
| case | finally | new | this |
| catch | float | package | throw |
| char | for | private | throws |
| class | goto* | protected | transient |
| const* | if | public | try |
| continue | implements | return | void |
| default | import | short | volatile |
| do | instanceof | static | while |

- . Son palabras reservadas en C:

| | | |
|---|---|---|
| auto | extern | sizeof |
| break | float | static |
| case | for | struct |
| char | goto | switch |
| const | if | typedef |
| continue | int | union |
| default | long | unsigned |
| do | register | void |
| double | return | volatile |
| else | short | while |
| enum | signed | |

# 4 Variables y constantes. Datos

- ***Variables***

Una variable es una localización o casillero en la memoria de un programa u ordenador con un nombre simbólico asociado a dicho espacio, y cuyo contenido cambia durante la fase de procesamiento de información. Son pues "contenedores de datos", y por ello se diferencian según el tipo de dato que son capaces de almacenar.

El **nombre de una variable** comenzará siempre por una letra, pudiendo contener a continuación tanto letras como números.

Las letras pueden ser tanto mayúsculas como minúsculas. No se admiten nombres de variables incluyendo espacios en blanco ni símbolos especiales como guiones, puntos, comas, comillas, etc. ni símbolos matemáticos ni palabras reservadas (que veremos más adelante). El nombre de una variable será lo suficientemente largo como para impedir que pueda confundirse con otra variable por tener nombre similar, así como para aportar una indicación de cuál es el contenido o función que cumple.

**Ejemplos:** *Nombre de variables.*

| Nombre de variable | Comentarios |
|---|---|
| Numerodeplantas | Válido, descriptivo |
| Importe | Válido, descriptivo |
| A | Válido pero no aporta información del contenido o función |
| AMC12 | Válido |
| AM12C | Válido |
| Coches usados | No válido (incluye un espacio) |
| Cochesusados ó CU | No válido (una variable tiene un único nombre) |
| Coches>30CV | No válido (incluye símbolo >) |
| Probabilidaddeaccidenteenbasealosdatosconocidos | Válido, pero no recomendable por ser excesivamente largo |
| Coches,motos | No válido (incluye una coma) |

Podríamos **clasificar** las variables según **distintos criterios**:

*** Variables según el tipo de dato que almacenan:**

- ***Variables Numéricas (Enteras y Reales)***
  - ***Variables de datos enteros (tipo Int)***: almacenan valores con números enteros.
  - ***Variables de datos reales (tipo Float)***: almacenan números decimales e irracionales usando la notación científica.

- **Variables Alfanuméricas**
  - ***Variables para datos de tipo carácter (tipo Chart)***: almacenan un único carácter (letra, signo o número)
  - ***Variables para datos de tipo cadena (tipo String)*:** almacenan un conjunto de caracteres.

- ***Variables booleanas o lógicas***: pueden almacenar sólo valores de lógica binaria (dos estados, “true” o “false)

Estos tipos de variables pueden tomar únicamente datos del mismo tipo, es decir, si la variable es entera (palabra reservada Int) solamente puede almacenar datos enteros, si la variable es declarada como cadena (palabra reservada String), sólo puede tomar valores correspondientes a ese tipo, etc

*** Variables según su ámbito:**

- ***Globales***: variables que pueden ser utilizadas a lo largo de todo el programa.
- ***Locales***: Variables que solo pueden ser utilizadas en la función o grupo de instrucciones donde éstas se declaran .

## • *Constantes*

Una constante es un **dato numérico o alfanumérico** que no cambia durante todo el desarrollo del algoritmo o durante la ejecución del programa. Es un objeto de **valor invariable**. Para expresar una constante se escribe explícitamente su valor.

**Tipos de Constantes**:

Al igual que las variables, las constantes pueden ser, según el tipo de valor:

- ***Constantes Numéricas (Enteras y Reales)***
- ***Constantes Alfanuméricas***
- ***Constantes Lógicas (Boolenas)***

Las constantes pueden ser:

Constantes sin nombre: Es una expresión numérica donde se puede utilizar directamente el valor.

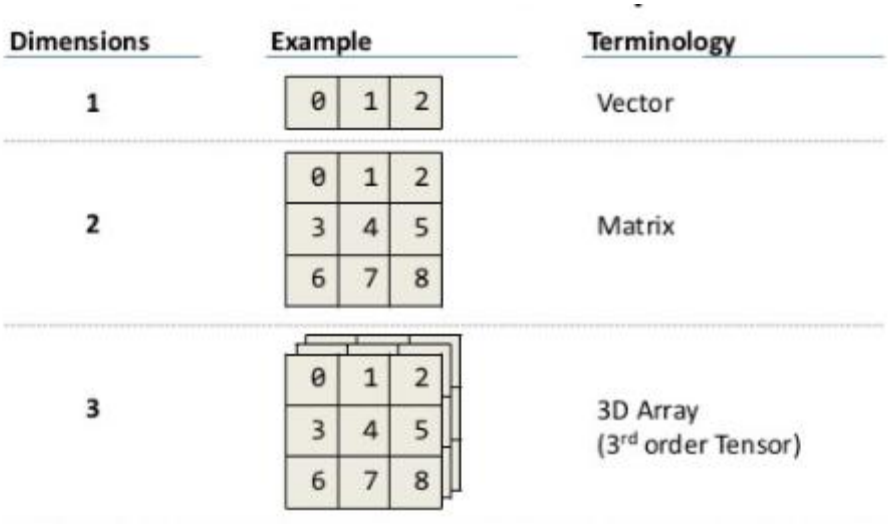
Constantes con nombre: Se hace una reserva de memoria en la cual se guarda el valor que será utilizado como constante.

## 5 Arrays

Un array es un medio de guardar un conjunto de elementos de la misma clase. Se accede a cada elemento individual del array mediante un número entero denominado índice. 0 es el índice del primer elemento y *n-1* es el índice del último elemento, siendo *n*, la dimensión del array. Así, el acceso a los elementos de los arrays se realiza a través de índices.

Los arrays son muy utilizados en la programación. Dependiendo de la cantidad de dimensiones que tengan pueden ser:

- ✓ De una dimensión (vectores). Por ejemplo $empleado[23]
- ✓ De dos dimensiones (matrices). Por ejemplo $butaca[3, 14]
- ✓ De tres o más dimensiones (multidimensionales). Por ejemplo $fecha[2012, 9, 22]




Las matrices, al igual que las variables, pueden almacenar no sólo números y valores, sino símbolos y caracteres. Podemos guardar valores de cualquier tipo de variable (string, entero, punto flotante, booleano).

Para acceder a los elementos del array se utilizan en muchos lenguajes de programación los corchetes [ ], dentro de los cuales existirá un localizador o índice que es un número entero.. Por ejemplo $empleado[23] podría tomar como valor $empleado[23] = “Juan Pérez Suárez”. En este caso se trataría de un array de cadenas de texto, es decir, un array de string. En otro caso empleado[23] podría tomar como valor empleado[23] = 2312. En este caso se trataría de un array de valores numéricos.

## 6. Comentarios

En el código fuente no sólo se pueden insertar comandos y sentencias que más tarde se ejecutarán, sino también comentarios. Un comentario es una parte de nuestro programa que el ordenador ignora y que, por tanto, no realiza ninguna tarea, no afectarán al programa.

Se utilizan generalmente para poner aclaraciones en lenguaje humano de lo que estamos haciendo en el lenguaje de programación cuando escribimos un programa. Ayudan a recordar al programador qué pretendía hacer en la línea o parte del programa donde está puesto el comentario, además de hacer que el código sea más comprensible para otras personas que

puedan acceder al código fuente. Son de gran ayuda para comprender el procedimiento seguido en un programa determinado. Para explicar, por ejemplo, qué guarda una variable o para qué sirve un fragmento de código, se utilizan comentarios.

Dependiendo del lenguaje de programación de que se trate, los comentarios van precedidos o entre determinados símbolos, por ejemplo *, //, ...

Ejemplos de comentarios:

```
# calcula el porcentaje de la hora que ha pasado
porcentaje = (minuto * 100) / 60
```

```
void loop()
{
  int valor=analogRead(A0); // declaro la variable valor como lectura del pin analógico A0, entrada del sensor de llama
if(valor>45)
{
```

```
void setup()
{
  size(500,500);//creamos una ventana de 500*500
  background(5,245,70);//damos color verde a la ventana

  stroke(0,0,0);// color negro del borde circunferencias

   strokeWeight(1); //grosor borde circunferencias 1,2, 3, 4, 5, 6, 7, 8, 9 y 10

   fill(255,255,255);// relleno blanco cicunferencias 1 y 2
   ellipse(250,250,400,400); //trazado circunferencia 1
   ellipse(250,250,360,360);// trazado circunferencia 2

    fill(0,0,0);// relleno negro cicunferencia 3
```

```
/*
 Este programa muestra el número de argumentos
 que se le han proporcionado a través de la línea de órdenes.
*/

int main(int argc, char * argv[])
{
 printf("El número de argumentos de este programa es %d\n", argc);
 return 0;
}
```

# 7. Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.

***ELEMENTOS DE UNA EXPRESION:***
- **Operandos**
- **Operadores**

Un **operador** es un símbolo o palabra que indica que se ha de realizar cierta acción entre dos o más valores, llamados **operandos.**

**CLASIFICACION DE LAS EXPRESIONES**
- **Expresiones Aritméticas**
- **Expresiones Lógicas**

## Expresiones ARITMÉTICAS

Son análogas a las fórmulas matemáticas. Las variables y constantes son numéricas (enteras o reales) y las operaciones son las aritméticas.

### OPERADORES MATEMÁTICOS.

En la siguiente tabla puedes ver los operadores matemáticos más frecuentes:

| OPERADOR | USO Y SIGNIFICADO | TIPOS DE | |
|---|---|---|---|
| | | OPERANDOS | RESULTADO |
| ^ | Eleva un valor a la potencia o exponente. | Entero o real | Entero o real |
| + | Suma dos valores. | Entero o real | Entero o real |
| - | Resta el valor de la derecha al de la izquierda. | Entero o real | Entero o real |
| * | Multiplica dos valores. | Entero o real | Entero o real |
| / | Divide el valor de la izquierda entre el de la derecha. | Real | Real |
| Div | División entera, devuelve un resultado de tipo entero. | Entero | Entero |
| Mod | Abreviatura de módulo, devuelve el resto de una división entera. | Entero | Entero |

**REGLAS DE PRIORIDAD O PRECEDENCIA**: las expresiones que tienen dos o más operandos requieren unas **reglas matemáticas que permitan determinar el orden de las operaciones**, se denominan reglas de prioridad o precedencia. Son las siguientes:

1. **Las operaciones que están encerradas entre paréntesis se evalúan primero**. Si existen paréntesis anidados (interiores unos a otros), las expresiones más internas se evalúan primero.

2. **Prioridad de Operadores Aritméticos**: dentro de una misma expresión, los operadores se evalúan en el siguiente orden:

| OPERADOR | ORDEN | OPERACION |
|---|---|---|
| ^ | Mayor | Potenciación |
| * , / | | Multiplicación y división |
| Div, Mod | | División entera y módulo |
| + , - | Menor | Suma y Resta |

3. En caso de **coincidir varios operadores de igual prioridad** en una expresión o sub-expresión encerrada entre paréntesis, el **orden de prioridad es de izquierda a derecha**.

## Expresiones LÓGICAS

Las Expresiones lógicas son aquellas que pueden tomar uno de dos valores:

verdadero (**V**) o falso. (**F**)

La importancia de estas expresiones es la aplicación en las estructuras de control que gobiernan el flujo de un programa. Las expresiones lógicas se forman combinando constantes y variables con operadores lógicos y relacionales.

## OPERADORES RELACIONALES.

Los operadores relacionales se utilizan para expresar condiciones. (comparación entre dos elementos)

| OPERADORES RELACIONALES | |
|---|---|
| **OPERADOR** | **SIGNIFICADO** |
| **=** | Igual que |
| **>** | Mayor que |
| **> =** | Mayor o igual que |
| **<** | Menor que |
| **< =** | Menor o igual que |
| **<>** | Diferente de |

El formato de una expresión lógica será:

| | | |
|---|---|---|
| **Expresión1 o valor** | **Operador de Relación** | **Expresión2 o valor** |

## OPERADORES LÓGICOS (puertas lógicas)

Son operadores que funcionan con valores booleanos (V ó F)

Las salidas o resultados de las operaciones con operadores lógicos se representan en forma de tabla. A estas tablas se las conoce con el con el nombre de **tablas de verdad**.

- **Operador NOT (no) o Negación: (Not p)**

| p | NOT p |
|---|---|
| V | F |
| F | V |

El operador o puerta lógica NOT devuelve la negada de la entrada: si entra el valor "verdadero" la salida es "falso" y si entra el valor "falso", la salida es "verdadero".

- **Operador AND (y) ó conjunción: (p and q)**

| p | q | p AND q |
|---|---|---|
| V | V | V |
| V | F | F |
| F | V | F |
| F | F | F |

La expresión p AND q, es verdadero solamente cuando ambas, p y q, son verdaderas, de lo contrario el resultado será falso.

- **Operador OR (o) ó Disyunción inclusiva: ( p OR q)**

| p | q | p OR q |
|---|---|---|
| V | V | V |
| V | F | V |
| F | V | V |
| F | F | F |

La expresión p OR q es falsa solamente cuando ambas expresiones son falsas, en caso contrario es verdadera.

## 8. Sentencias

Si bien las expresiones son unidades o componentes elementales, las sentencias son unidades completas, ejecutables en sí mismas, de rango superior a las expresiones.

Así, muchos tipos de sentencias incorporan expresiones aritméticas o lógicas como componentes de dichas sentencias

Las sentencias son los elementos básicos en que se divide el código de un programa de un determinado lenguaje de programación. Son las instrucciones u órdenes que se le dan al programa para realizar una tarea específica (mostrar un mensaje en la pantalla, declarar una variable, inicializarla, llamar a una función, etc.)

Las sentencias acaban con **;**. , este carácter separa una sentencia de la siguiente. Podemos decir que cualquier instrucción acabada en **;** forma una sentencia.

Existen casos en que no es necesario incluir el **;** , por ejemplo cuando después de la sentencia hay una palabra reservada (end, eles, until, ...).

Normalmente, las sentencias se ponen unas debajo de otras, aunque sentencias cortas pueden colocarse en una misma línea. Son ejemplos de sentencias:

```
int i=1;
import java.awt.*;
System.out.println("El primer programa");
rect.mover(10, 20);
```

De ahí que podemos decir que un programa es una secuencia de sentencias que se ejecutan para realizar una determinada tarea.

Las sentencias pueden ser de dos tipos: simples y estructuradas o de control

**Sentencias simples:** son una única instrucción; las sentencias simples son ejecutadas secuencialmente, una después de la otra.

Pueden ser:

- sentencia de salto incondicional (goto)
- sentencia de llamada a procedimiento
- sentencias de asignación

- **sentencia** de salto condicional GOTO

La sentencia ***goto*** se utilizaba mucho en los primeros lenguajes de programación porque era la única manera de saltar de una instrucción del programa a otra.

Se ha comentado algo sobre el *goto* por curiosidad e historia, pero olvídate de que existe. Es una mala herramienta de programación y se puede cambiar por las sentencias de control repetitivas y alternativas ( sentencias estructuradas)

- **sentencia** de llamada a procedimiento

Esta sentencia consiste en llamar a un procedimiento, y se hace poniendo el nombre del procedimiento seguido de un punto y coma.

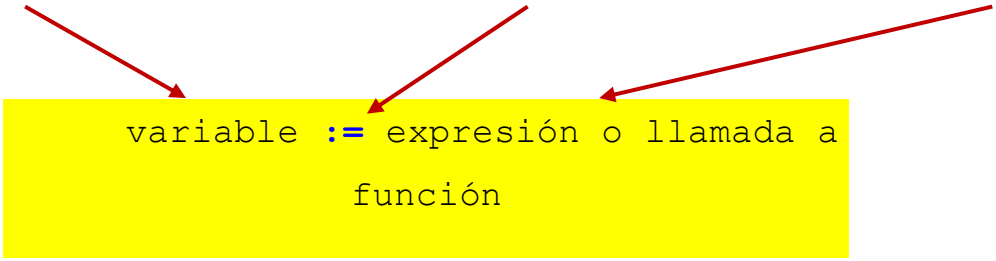
Un procedimiento es un subprograma que creas con el fin de realizar una cierta tarea, así llamarlo desde el programa principal en los sitios que se necesite. Es decir, se divide un programa grande en otros más pequeños para llamarlos cuando los necesites. A esto se le llama programación procedimental.

- **Sentencia** de asignación

Es una de las instrucciones más comunes en un programa. Permite darle un primer valor a una variable o cambiar su valor.

Una sentencia de asignación consta de tres partes:

la parte a **la izquierda** del símbolo de asignación, el propio **símbolo** de asignación, y la parte de **la derecha.**




Mientras que la parte de la izquierda sólo puede ser una variable, la de la derecha puede ser una expresión o una llamada a una función. En el siguiente ejemplo puedes ver unas cuantas asignaciones correctas:

```
  begin
            (* ... *)

    x := 10;            (* literal constante *)
    y := w;             (* "w" puede ser constante o variable
*)
    z := maximo(x,y);   (* llamada a una funcion *)
    valor_medio := (a + b) / 2;

            (* ... *)
  end.
```

Las sentencias estructuradas o de control permiten **controlar el flujo del programa, tomando decisiones a partir de comparaciones y generando bucles** mientras o hasta que se cumplan ciertas condiciones.

Controlar el flujo es **determinar el orden en el que se ejecutarán las instrucciones** en nuestros programas. Si no existiesen las sentencias de control entonces los programas se ejecutarían de forma secuencial, empezarían por la primera instrucción e irían una a una hasta llegar a la última.

Con las sentencias de control tenemos la posibilidad de **elegir** uno de entre varios caminos en función de ciertas condiciones (**sentencias alternativas** o selectivas: if, case, ...). Y por el otro, no se podría ejecutar algo repetidas veces, sin tener que escribir el código para cada una (**sentencias repetitivas** o iterativas: for, while, repeat, ...).

## 9. Funciones

Una función es un **conjunto de líneas de código que realizan una tarea determinada.** Se utilizan para descomponer grandes problemas en tareas simples y para ejecutar operaciones que son comúnmente utilizadas durante un programa y de esta manera reducir las líneas de código. Una **función** es una sección de un programa que calcula un valor de manera independiente al resto del programa.

Una función tiene tres componentes importantes:

- los **parámetros**, que son los valores que recibe la función como entrada;
- el **código de la función**, que son las operaciones que hace la función; y
- el **resultado** (o **valor de retorno**), que es el valor final que entrega la función.

En esencia, una función es un mini programa. Sus tres componentes son análogos a la entrada, el proceso y la salida de un programa.

**Cuando una función es "llamada", se ejecuta el código en su interior,** y una vez que ésta finalice, el programa regresa al punto desde el cual la función fue llamada.