

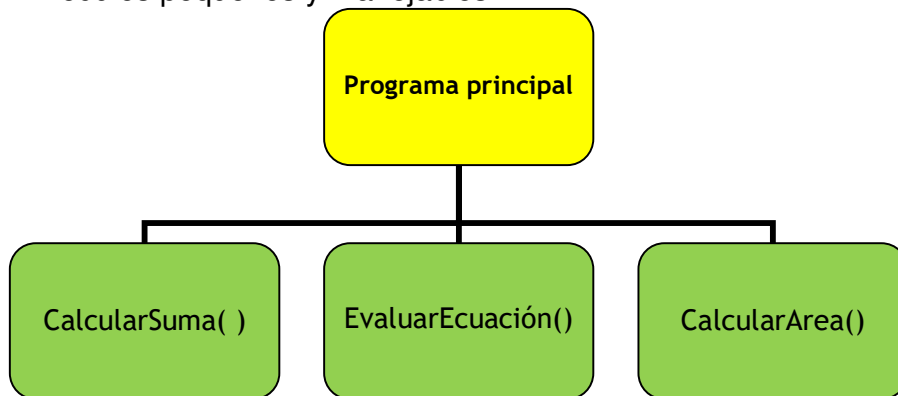
Capítulo IV. FUNCIONES

Introducción

Las funciones son el conjunto de bloques de instrucciones que realizan tareas bien definidas.

Sólo devuelve un valor o ninguno.

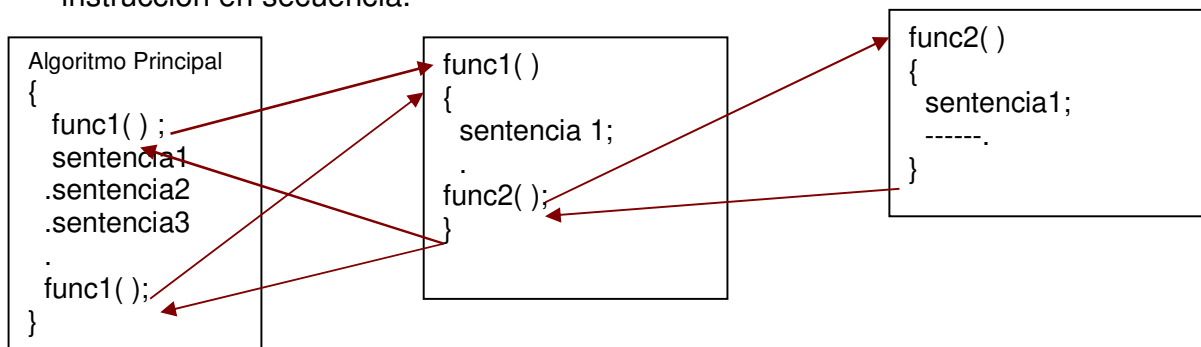
Las funciones ahorran espacio, reducen repeticiones y hacen más fácil la programación, al proporcionar un medio de dividir un problema muy grande en módulos pequeños y manejables.



Todo algoritmo, puede manejar tantas funciones como se requiera, pero el mismo *inicia* su ejecución *desde* el programa principal.

Cómo se ejecuta una función ?

✚ Cuando una función se llama, el control se pasa a la misma para su ejecución, se devuelve el control al módulo que llamó y se continúa con la siguiente instrucción en secuencia.



Para qué se usan las funciones?

- Se usan para realizar tareas concretas y simplificar el programa
- sirven para evitar escribir el mismo código varias veces.

Tienen las siguientes características:

- Generan respuesta a una llamada
- Procesan datos
- Corresponden a código ejecutable (instrucciones o sentencias)
- Sólo debe retornar un valor (si retorna)

Ventajas de utilizar funciones:

1. Permite trabajar en módulos, ya que se dividen las tareas
2. Reutilización de código
3. Programas más fáciles de mantener (más legibles y más cortos)
4. Administración de memoria (ocupa menos espacio)

Funciones más conocidas :

Existen algunas funciones incorporadas en las calculadoras tales como : sin, cos, tan , sqrt, abs.

Nosotros trabajaremos las funciones “**Definidas por el usuario**”, por lo cual es importante conocer la sintaxis a utilizar y cómo se maneja.

1. ESTRUCTURA DE UNA FUNCIÓN.

Las funciones tienen tres componentes principales:

- a. Encabezado ó Definición de la función
- b. Variables locales
- c. Cuerpo de la función

a. ENCABEZADO O DEFINICIÓN DE UNA FUNCIÓN:

Declara el nombre de la función y la lista de parámetros formales con la cual se va a trabajar. También identifica el tipo de dato que retorna la función.

Formato:

Tipo de dato nombre_de_la Función (lista de parámetros formales)

Donde:

Tipo: especifica qué tipo de dato que retorna la función. Este puede ser cualquier tipo básico: entero, flotante, carácter, cadena.

Este valor es devuelto a la sentencia de llamada, por medio de la instrucción **retornar** (el cual puede ser una variable o una expresión).

Nombre de la Función: identificador que indica el nombre de la función. Se escribe iniciando con un verbo y usando camelCase.

Ejemplos: procesarDatos, mostrar_Tabla1, cargarMatriz

Se recomienda utilizar nombres cónsonos con la tarea que va a realizar la función.

Lista de Parámetros Formales: son las variables que reciben los valores desde el algoritmo principal, los cuales se envían cuando se llama a la función (conocidos como *parámetros actuales*). Los valores que se suministran a la función pueden ser variables y/o constantes.

Los parámetros formales deben ser declarados con su tipo de dato correspondiente, se separan con coma y son locales a la función, lo cual significa que serán reconocidos sólo en la función donde están declarados.

A continuación, ejemplos de definiciones de funciones ó encabezados con parámetros formales :

Ejemplo 1:

Se define la función suma que acepta un entero y un flotante y retorna un valor entero.

```
entero sumarV (entero dato1, flotante dato2) ← Definición de la función
{
    retornar valor; }
```

Ejemplo 2:

Se define la función calcularDeriv que no tiene argumentos y retorna un valor real

```
flotante calcularDeriv ( ) {
    retornar valor; }
```

Ejemplo 3:

Se define la función dibujarCono que no tiene parámetros formales y no retorna valor.

```
dibujarCono ( ) {
}
```

2. Variables locales a la función

Son las variables declaradas por el usuario dentro de la función. Todas las variables aquí declaradas son de uso exclusivo de la función y **no son reconocidas fuera de ella**.

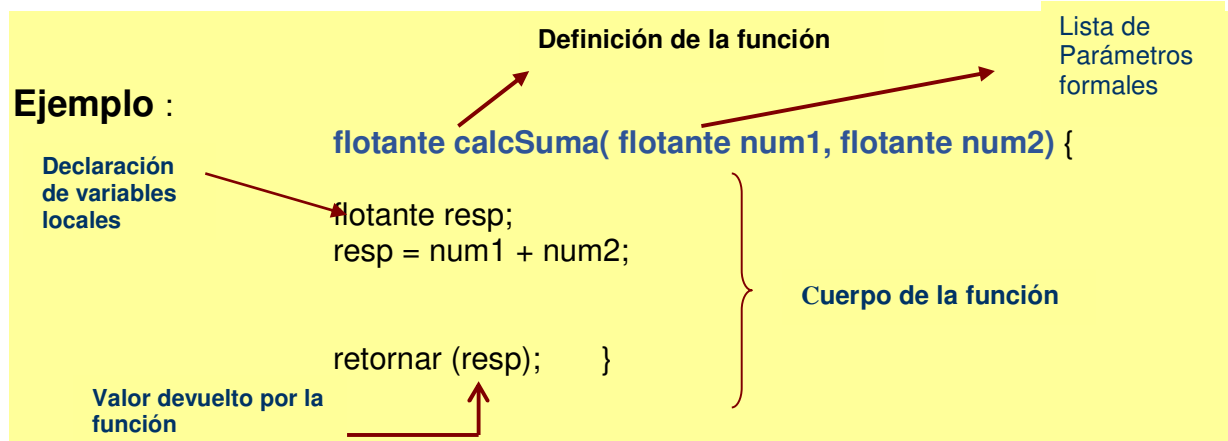
Al igual que en el programa principal, en una función pueden declararse variables, constantes, arreglos.

3. Cuerpo de la función:

Es el conjunto de instrucciones en secuencia lógica que componen la función y que realizarán la tarea solicitada.

Todas las instrucciones se encierran entre corchetes de **apertura** { } y **cierre**.

Ejemplo de una función con sus componentes:



- Si la función retorna un valor debe aparecer **una sentencia de retorno**, la cual presenta el siguiente formato:

Formato : **retornar (expresión) ;**

Una función puede tener cualquier número de sentencias retornar. En el momento en que el programa encuentre la primera, se retorna al punto de llamada.

Si la función no retorna un valor, se omite la sentencia retornar, y no se coloca tipo en la definición de la función.

EJEMPLO: **calcularTotal(entero a, entero b)**{
 flotante x;
 sentencia 1;
 sentencia n; }

EJEMPLO: buscar el mayor de tres.

```
Algoritmo buscarMayor{
    entero a, b, c;
    entero may
    may = Buscar_Mayor (a, b, c) //llamada a la función
    ("Valor mayor es:", may );
}

entero BuscarMayor( entero x, entero y, entero z) //definición de la función
{
    entero max
    max= x;
    if ( y > max)
        max = y
    if ( z > max)
        max = z
    retornar(max)
}
```

▪ LLAMADA A LA FUNCIÓN

Para que una función se ejecute ésta tiene que ser llamada. Esto hará que se envíe el control del programa a la función nombrada. Se puede llamar a una función especificando su nombre, seguido de una lista de argumentos encerrados entre paréntesis llamados parámetros actuales y separados por comas.

Formatos de llamadas a funciones:

Nombre de identificador = nombre de la función(parámetros actuales);

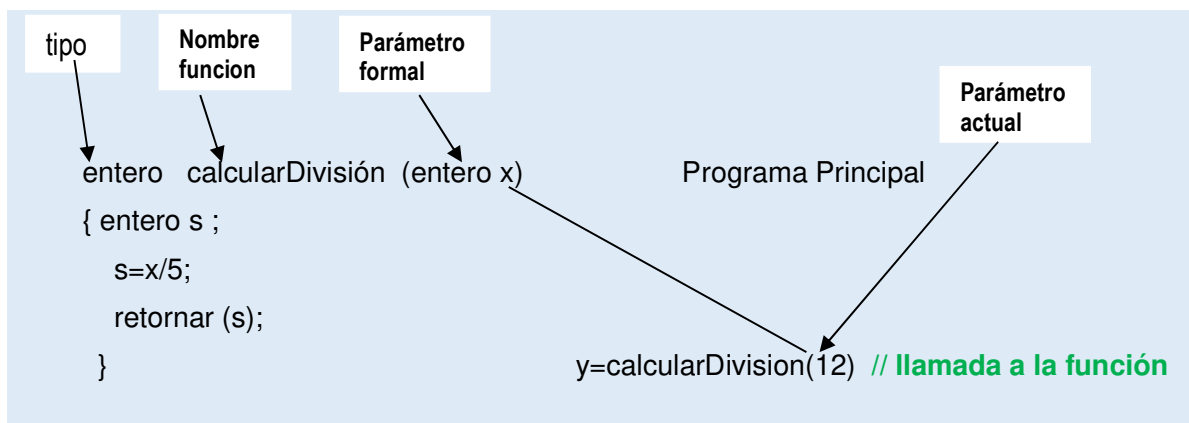
Nombre de identificador = nombre de la función();

nombre de la función(lista de parámetros actuales);

Ejemplos:

```
r = multiplicar (a, b); // llamada a la función multiplicar y devuelve el valor a r
sumar (a, b * 2);      // llamada a la función suma, no devuelve valor
```

Ejemplo del cuerpo de una función y la llamada a la función:



En la función `calcularDivisión`, la variable `x` recibe el valor de `12` que es pasado en la llamada de la función, por tanto `x=12` y se procede a dividir entre 5, una vez que se ejecuta la división, se devuelve (retornar) el contenido de la **variable s**. En el programa principal, el resultado de la división (`12/5`) lo recibe **y**.

Los parámetros permiten la comunicación de la función con el resto del programa mediante transferencia de datos.

En una llamada normal a una función, habrá un parámetro actual por cada parámetro formal. Los parámetros actuales pueden ser constantes, variables simples o expresiones.

Habrà un argumento actual por cada argumento formal y deben coincidir en :

- ✓ Cantidad
- ✓ Tipo de dato
- ✓ Valor posicional que ocupa cada argumento.

EJEMPLO: Hacer un algoritmo que lea dos enteros y llame a la función `sumar` que realice el cálculo e imprima el resultado en el programa principal.

Algoritmo Sumar

```

{
  entero a, b, c;
  imprimir("Introduzca dos enteros");
  leer(a,b);
  // llamada a la función
  c=sumar(a, b) ; //c toma el valor que trae la función
  imprimir ("La suma de ", a,"+", b, "es", c);
}

//función
entero sumarV(entero x, entero y) //definición
{ entero suma;
  suma = x+y;
  retornar (z);
}

```

4. PASO DE ARGUMENTOS A UNA FUNCIÓN:

Existen dos métodos para pasar argumentos a una función, por valor y por referencia. Sin embargo, puede que la función no maneje ningún parámetro es decir, no reciba parámetros.

El **PASO POR VALOR** mediante un argumento actual, ***copia el valor del argumento actual al argumento formal de la función***. Si se cambia el contenido de un parámetro formal, el cambio solo se refleja en la función donde se hace el cambio y no tiene efecto fuera de la misma.

En este tipo de proceso, la función receptora no puede modificar los contenidos de las variables que recibe. Este procedimiento se conoce como ***paso por valor***.

EJEMPLO:

```
modificarV (entero a)
{
    a = a * 3
    imprimir ("a = ", a);
    retornar;
}
```

```
Algoritmo Dato {
    entero a;
    a = 2;
    imprimir(" = ", a)
    modificarV (a)
    imprimir (" a = ", a)
}
```

Salida :

Antes : 2

Durante(función) : 6

Después : 2

Consideraciones Finales:

1. Una función no debería tener una longitud mayor que una página (Reutilización del software).

2. Los programas deberían ser escritos como recopilaciones de pequeñas funciones. Esto haría los programas más fáciles de escribir, depurar, mantener y modificar.
3. una función que requiere un gran número de parámetros quizás está ejecutando demasiadas tareas. Considere dividirla en funciones más pequeñas.
4. La definición de la función y las llamadas de función deberán estar de acuerdo a lo que se refiere al número, tipo y orden de los argumentos o parámetros, así como en el tipo de valor de regreso.

Ejemplo de Ejecución de una función:

Llamada a la Función

Algoritmo Principal

```

{
    entero num1, num2, sum;
    imprimir("Introduzca dos enteros");
    leer(num1, num2);
    sum = sumar(num1, num2);
    imprimir("La suma de ", num1, "+", num2, "es", sum);
}
        
```

amada a la función

//función

```

entero sumar(entero n1, entero n2) //definición
{
    entero suma;
    suma = n1 + n2;
    retornar(suma);
}
        
```

1. Una vez el control se pasa a la función **sumar** los parámetros formales reciben **una copia** de los parámetros actuales. Es decir, ahora **trabajamos** en el espacio de memoria de la función.

Memoria Algoritmo principal		Memoria Función	
num1		n1	
num2		n2	
sum		suma	

Salida:

Prof. Mitzi Murillo de Velásquez Msc.

6

Llamada a la Función

Algoritmo Principal

```

{
    entero num1, num2, sum;
    imprimir("Introduzca dos enteros");
    leer(num1, num2);
    sum = sumar(num1, num2);
    imprimir("La suma de ", num1, "+", num2, "es", sum);
}
        
```

amada a la función

//función

```

entero sumar(entero n1, entero n2) //definición
{
    entero suma;
    suma = n1 + n2;
    retornar(suma);
}
        
```

2. Realizado los procesos en la función, el control **se devuelve o retorna** (retornar suma) al programa principal y su contenido se asigna a la variable **sum** en el programa principal.

Memoria Algoritmo principal		Memoria Función	
num1	9	n1	9
num2	15	n2	15
sum		suma	24

Salida:

Prof. Mitzi Murillo de Velásquez Msc.

Ejecución de la Función

Algoritmo Principal

```

{
    entero num1, num2, sum;
    imprimir("Introduzca dos enteros");
    leer(num1, num2);
    sum = sumar (num1, num2);
    imprimir ("La suma de ", num1, "+", num2, "es", sum);
}
        
```


24 amada a la función
9
15

3. Al devolverse el control al programa principal las variables de memoria en la función pierden su valor.

//función sumar

```

entero sumar(entero n1, entero n2) //definición
{
    entero suma;
    suma = n1 + n2;
    retornar (suma);
}
        
```



Memoria Algoritmo principal		Memoria Función	
num1	9	n1	—9
num2	15	n2	—15
sum	24	suma	—24

Salida: La suma de 9 + 15 es 24

Prof. Mitzi Murillo de Velásquez Msc.

Ejemplo Completo

Ejemplo no. 1

Calcular el área de un triángulo cuya fórmula es $\frac{1}{2} * b * h$. Mostrar el área.

Algoritmo AreaT{

```

// Declarativas
flotante base, altura, area;
//cuerpo
leer( base, altura);
area = CalcArea(base, altura);
imprimir("Area del triangulo:", area);
}

entero CalcArea(flotante b, flotante h)
{
    //declarativas locales
    flotante ar;
    // cuerpo
    ar = (b * h)/2;
    retornar(max);
}
        
```

Entrada	El área (área) y la base (base)
Proceso	Llamar a la función AreaT() Calcular área del triángulo
Salida	area

Memoria Algoritmo principal		Memoria Función	
base		b	
altura		h	
area		ar	

Prof. Mitzi Murillo de Velásquez Msc.