Universidad Continental

FUNDAMENTOS DE PROGRAMACIÓN

¿Qué aprendimos la sesión anterior?

Programas utilizando estructura repetitiva

1. Realice un programa que muestre y sume los números múltiplos de 7 que existen en la serie de 158 a 362, utilizando while y for.
2. Realice un programa que muestre y sume la siguiente serie, donde N es un número entero ingresado por teclado:

$$\frac{3}{1} + \frac{3}{2} + \frac{3}{3} + \frac{3}{4} + \cdots + \frac{3}{N}$$

3. Realice un programa que muestre y sume la siguiente serie, donde N es un número entero ingresado por teclado:

$$3^0 + 3^1 + 3^2 + 3^3 + \ldots + 3^N$$

MÓDULOS PARA LA PROGRAMACIÓN

Función y Procedimiento

Semana 09

Propósito

Reconoce los tipos de módulos tales como funciones y procedimientos en la programación con C++.

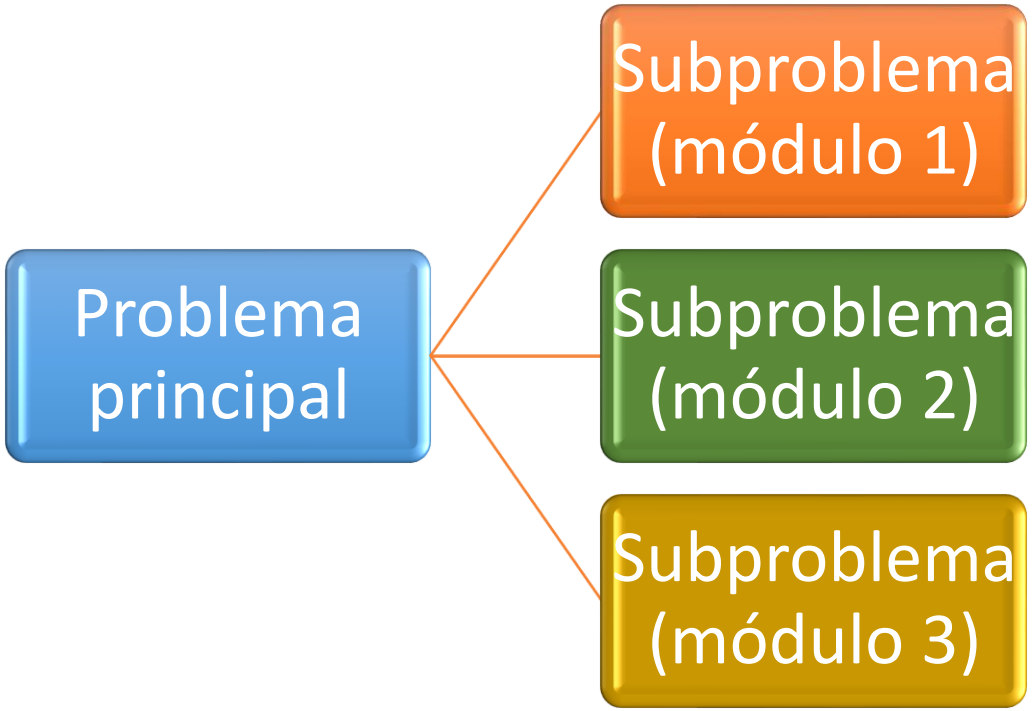
Agenda del día

1. Modularización de programas: definición y características
2. Paso de parámetros en los módulos de programa
3. Función
4. Procedimiento
5. Función VS Procedimiento

Modularización de programas: definición y características

Definición de Módulo de Programa

- El módulo de programa es un segmento o porción de código, ***independiente y reutilizable,*** para el mismo programa o para otros programas.
- Es decir, un módulo puede contener el desarrollo de un algoritmo y ser usado más de una vez.



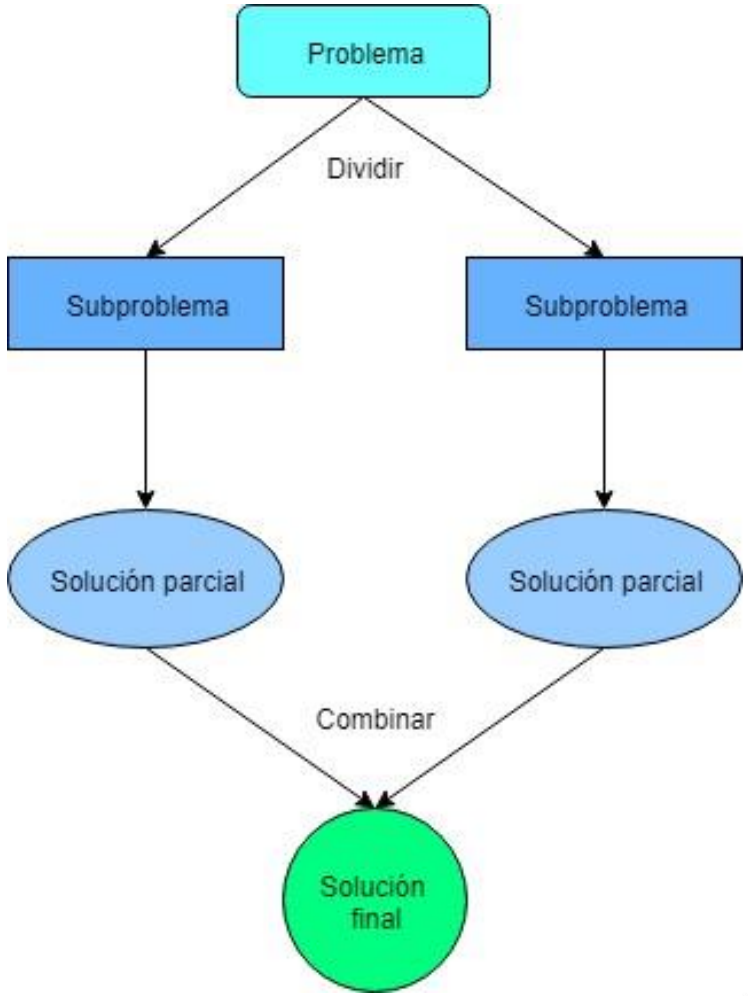
Característica de la Modularización de programas

- Diseños descendente (Top-down)

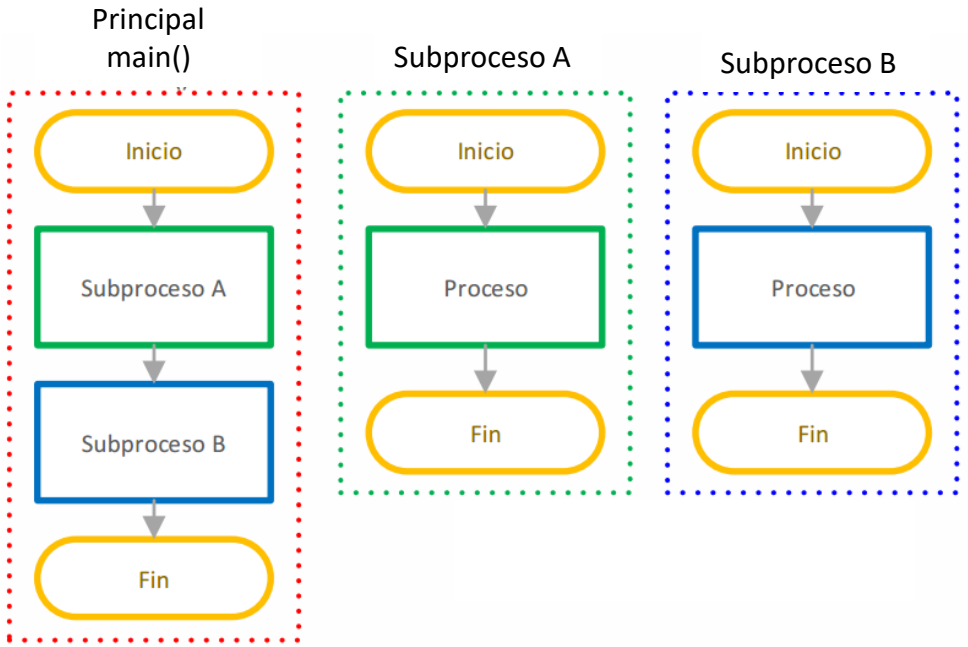
  Dividir el problema en subproblemas más pequeños

- Módulos independientes entre sí

- Se puede reusar el código



Ejemplo modularización




```
#include<iostream>
using namespace std;

Modulo1( )
{  ..............
   ..............
}

Modulo2( )
{  ..............
   Modulo1();
   ..............
}

void main( )
{
   ..............
   ..............
   ..............
 Modulo2();
   ..............
   ..............
 Modulo1();
   ..............
   ..............
}
```

```
#include<iostream>
using namespace std;

Modulo3( )
{  ..............
   ..............
}

void main( )
{
   ..............
   ..............
   ..............
 Modulo1();
   ..............
   ..............
 Modulo3();
   ..............
   ..............
}
```

Tipos de módulos o subprogramas

- Todos los lenguajes de programación admiten subprogramas o módulos
- Se los denomina **funciones, procedimientos**, subrutinas.

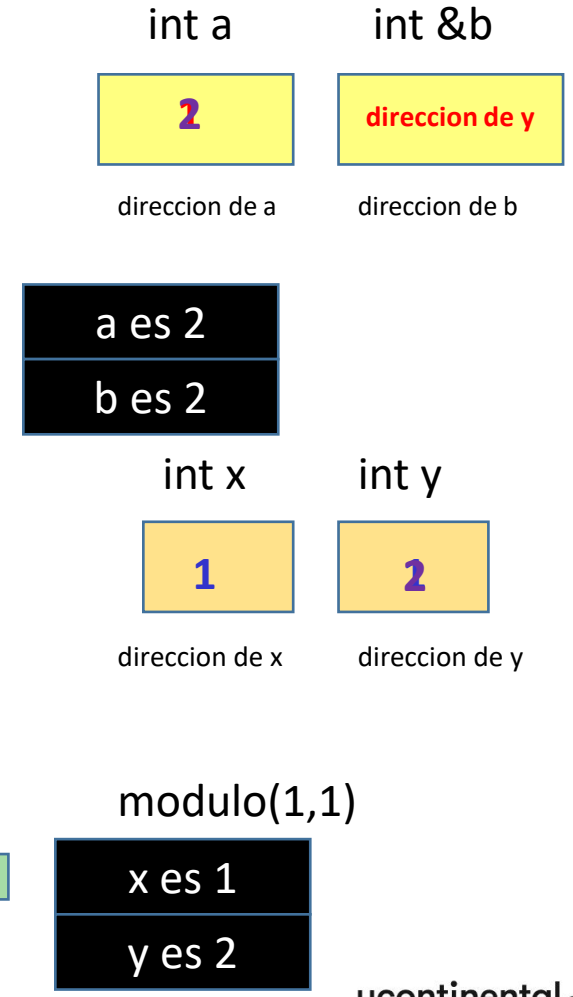
Paso de parámetros en los módulos de programa

Ejemplo de paso de parámetros por valor y por referencia

La variable "a"
paso de parámetros por valor.

La variable "b"
paso de parámetros por referencia.

```
#include<iostream>
using namespace std;

void  modulo(int a, int &b)
{
    a++;
    b++;
    cout<<"El valor de a es: "<<a;
    cout<<"El valor de b es: "<<b;
}

void main()
{   int x, y;

    x=1;
    y=1;
    modulo(x,y);
    cout<<"El valor de x es: "<<x;
    cout<<"El valor de y es: "<<y;
}
```

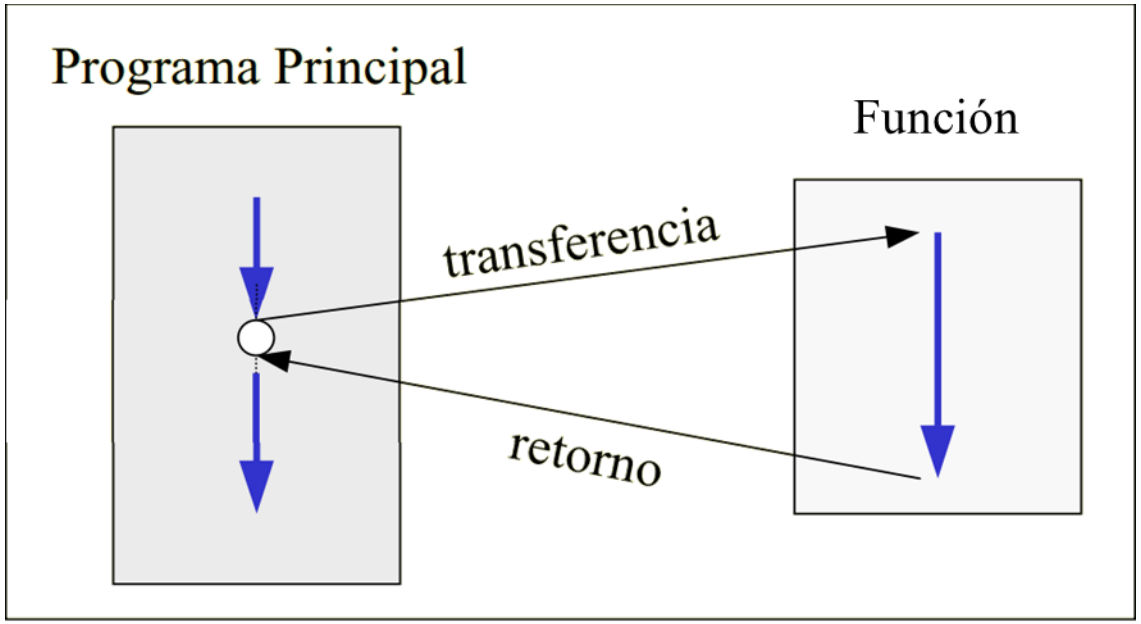


Función

Función en C++

Una función es un conjunto de líneas de código que realizan una tarea específica y puede **retornar un valor**.

Las funciones pueden tomar **parámetros** que modifiquen su funcionamiento.



Sintaxis:

A la declaración de una función se le denomina PROTOTIPO

```
//Declaración de la función
tipo_resultado  nombre_función (lista_de_parámetros);

//Definición
tipo_resultado  nombre_función (lista_de_parámetros)
{
        cuerpo_de_la_función;
        return expresión; //la expresión es de tipo tipo_resultado
}
```

Donde:

tipo_resultado: Es el tipo de dato que devuelve la función

expresión: Valor que devuelve la función

lista_de_parámetros: aparecen con su tipo, la función usa estos valores en el cuerpo.

Declaración y definición de funciones

- En C++ hay que distinguir entre lo que es un declaración y una definición de una función.

- En la declaración de una función tan sólo se incluye la cabecera o **prototipo** de la misma y tiene que aparecer antes de ser utilizada.

- En la **definición** de la función aparecen las sentencias que ejecuta dicha función, y puede aparecer en cualquier parte del programa.

Funciones en C++

Prototipos , tipos de resultados y argumentos.

```
float vol_cilindro(float r, float a);

float promedio3(int nota1,int bono,int suma);

int suma(int a, int b);

double aporte(int a,int b,int c,float X1,float X2);

char strcat(char c1, char c2);
```

Sobrecarga de funciones:

```
1   //Sobrecarga de la función cubo
2   #include<iostream>
3   using namespace std;
4
5   int cubo(int);
6   float cubo(float);
7
8   int main()
9   {
10      int ne;
11      float nr;
12      cout<<"Ingrese numero entero: ";
13      cin>>ne;
14      cout<<"Ingrese numero real: ";
15      cin>>nr;
16      cout<<"El cubo de "<<ne<<" es "<<cubo(ne)<<endl;
17      //el compilador al detectar un valor int, elige esta función
18      cout<<"El cubo de "<<nr<<" es "<<cubo(nr)<<endl;
19      //la llamada es igual con un dato flotante
20      cout<<endl<<endl;
21      return 0;
22  }
23  int cubo(int c) //función sobrecargada para valores enteros
24  {
25      return c*c*c;
26  }
27  float cubo(float c)  //función sobrecargada para valores flotantes
28  {
29      return c*c*c;
30  }
```

Ejemplo 1 – Cuadrado de un número

Para comenzar, vamos a considerar el caso en el cual se desea crear la función **Cuadrado()**, que deberá devolver el cuadrado de un número real (de punto flotante), es decir, **Cuadrado()** aceptará números de punto flotante y regresará una respuesta como número flotante.

Ejemplo – Cuadrado()

```
 4  float Cuadrado(float num); //Prototipo

18  float Cuadrado(float num) //Definición
19  {
20      return num*num;
21  }
```

Nota: aunque para la función que veremos el tipo de retorno coincide con el tipo de parámetro pasado, algunas veces las cosas pueden cambiar, es decir, no es obligatorio que una función reciba un parámetro de un tipo y que tenga que regresar una respuesta de dicho tipo.

Función que retorna el cuadrado de un número:

```
1  // regresar el cuadrado de un número
2  #include<iostream>
3  using namespace std;
4
5  double Cuadrado(double n);      //Prototipo
6
7  int main()
8  {
9      double base;
10     cout<<"Ingrese número a elevear al cuadrado: "<<endl;
11     cin>>base;
12     cout<<Cuadrado(base);       //Llamada a función
13     return 0;
14 }
15
16 double Cuadrado(double n)       //Definición
17 {
18     return n*n;
19 }
```

Ejemplo 2 – Factorial

Crear la función **factorial()**, que deberá devolver el factorial de un número entero, es decir, **factorial()** aceptará un número entero y retornará su factorial como como número flotante.

Función que retorna el factorial de un número:

```
1  #include<iostream>
2  using namespace std;
3
4  float factorial(int); //Prototipo o declaración de la función
5
6  int main()
7  {
8      int num;
9      cout<<"Ingrese un numero para hallar su factorial: ";
10     cin>>num;
11     cout<<"El factorial de "<<num<<" es ";
12     cout<<factorial(num); //Llamado a la función
13     cout<<endl<<endl;
14     system("pause");
15     return 0;
16 }
17
18 float factorial(int N) //Definición
19 {
20     float F=1;
21     for(int i=1; i<=N; i++)
22         F=F*i;
23     return F;
24 }
```

Ejercicios

A. Escribe un programa que implemente y utilice una función para determinar si un número es positivo o negativo. Lee un número entero por teclado e imprime por pantalla si el número leído es positivo o negativo haciendo uso de la función definida.

B. Realizar una función llamada Promedio, que calcula el promedio de tres números enteros y retorna el resultado en un número real.

Problema A

```
1  #include<iostream>
2  using namespace std;
3  int posinega(int); //Prototipo de la función
4  int main()
5  {
6      int num;
7      cout<<"Ingrese numero: ";
8      cin>>num;
9      if (posinega(num)==1)
10         cout<<"Es positivo o neutro \n";
11     else
12         cout<<"Es negativo \n";
13     cout<<endl<<endl;
14     return 0;
15 }
16 int posinega(int N) //Definición de la función
17 {
18     if (N>=0)
19         return 1;
20     else
21         return 0;
22 }
```

Problema B

```cpp
#include<iostream>
using namespace std;

float Promedio(int, int, int); //Prototipo de la función

int main()
{
    int A,B,C;
    cout<<"Ingrese tres numeros enteros: \n";
    cin>>A>>B>>C;
    cout<<"El promedio es "<<Promedio(A,B,C); //Llamado a la función
    cout<<endl<<endl;
    return 0;
}
float Promedio(int x, int y, int z) //Definición de la función
{
    return (x+y+z)/3.0;
}
```

Procedimiento

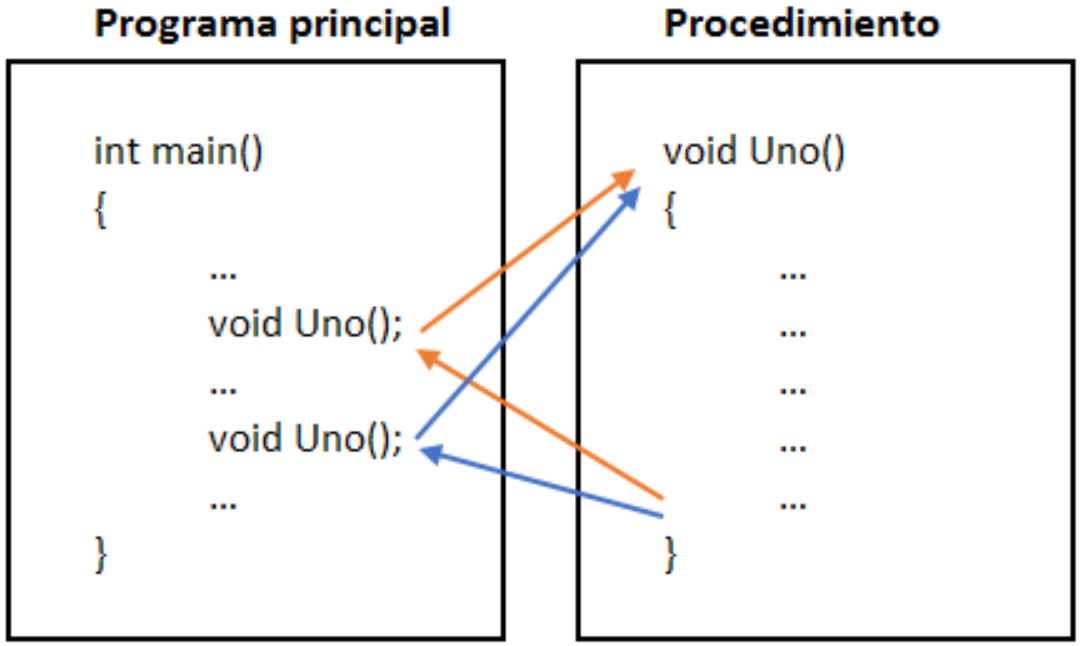
¿Qué son los procedimientos?

Aunque las funciones son herramientas de programación muy útiles para la resolución de problemas, su alcance está muy limitado.

Con frecuencia se requieren subprogramas que calculen varios resultados en lugar de uno solo. En estas situaciones la función no es apropiada y se necesita disponer del otro tipo de subprograma: el procedimiento o subrutina.

¿Qué son los procedimientos?

- Un procedimiento o subrutina es un subprograma que ejecuta un proceso específico.
- Cuando se invoca el procedimiento, los pasos que lo definen se ejecutan y a continuación se devuelve el control al programa que le llamó.



Sintaxis:

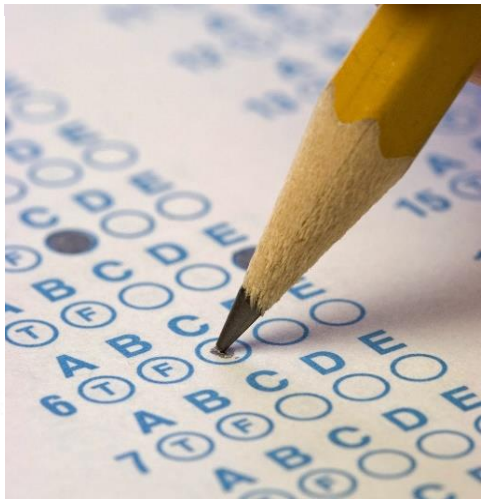
```
//Prototipo o Declaración del procedimiento
void  nombre_procedimiento (lista_de_parámetros);


//Definición
void  nombre_procedimiento (lista_de_parámetros)
{
      sentencias del procedimiento (sin valor de retorno)
}
```

Nota:

Si el procedimiento o la función se define sobre el programa principal, no es necesaria su declaración o prototipo.

Ejemplo

- Elaborar un procedimiento que reciba como parámetro una nota (número entero) y verifique si la nota es correcta; de ser así también indique si es aprobada o desaprobada. Luego escribir el programa principal que invoque a este procedimiento.



Con prototipo

```cpp
#include<iostream>
using namespace std;

void nota (int); //Prototipo o declaración del procedimiento

int main()
{
    int N;
    cout<<"Ingrese una nota (0-20): ";
    cin>>N;
    nota (N); //Llamado al procedimiento
    cout<<endl<<endl;
    system("pause");
    return 0;
}

void nota (int A) //Definición del procedimiento
{
    if (A<0 || A>20)
        cout<<"Nota incorrecta \n";
    else
        if (A<11)
            cout<<"Desaprobado \n";
        else
            cout<<"Aprobado \n";
}
```

Sin prototipo

```cpp
#include<iostream>
using namespace std;

void nota (int A)   //Prototipo y definición del procedimiento
{
    if (A<0 || A>20)
        cout<<"Nota incorrecta \n";
    else
        if (A<11)
            cout<<"Desaprobado \n";
        else
            cout<<"Aprobado \n";
}

int main()
{
    int N;
    cout<<"Ingrese una nota (0-20): ";
    cin>>N;
    nota (N);       //Llamado al procedimiento
    cout<<endl<<endl;
    system("pause");
    return 0;
}
```

Función VS Procedimiento

Procedimientos vs Funciones

- Los procedimientos y funciones son subprogramas cuyo diseño y misión son similares; sin embargo, existen unas diferencias esenciales entre ellos:

  - Las funciones normalmente devuelven un único valor a la unidad de programa que la llama. (**return**)
  - Los procedimientos suelen devolver más de un valor, o pueden no devolver ninguno si solamente realizan alguna tarea, como mostrar un mensaje o una operación de salida.
  - Los procedimientos se pueden declarar y definir antes del programa principal.

Preguntas

Reflexionemos

ucontinental.edu.pe